Fraunhofer Institut Experimentelles Software Engineering

The Library Systems Product Line

A KobrA Case Study

Authors:
Joachim Bayer
Dirk Muthig
Brigitte Göpfert

IESE-Report No. 024.01/E
Version 1.0
November 22, 2001

A publication by Fraunhofer IESE

Executive Summary

This report presents a case study in the domain of library and information systems that accompanied the KobrA method definition to illustrate the method's concepts and to experiement with alternative ideas.

Keywords: KobrA, Product Line Engineering

Table of Contents

1 Introduction

The KobrA method, which has been developed at the Fraunhofer Institute for Experimental Software Engineering (IESE), provides a common and integrated viewpoint on several IESE competencies under the umbrella of a systematic method for component-based product line engineering. This report presents the case study in the domain of library and information systems that accompanied the KobrA method definition to illustrate the method's concepts and to experiement with alternative ideas. More information on the background of the case study is given in Section 1.1.

Section 1.2, then, ellaborates on the objectives of the case study project and on the reasons for spending effort on an academic case study instead of taking material from the context of our transfer projects in industrial contexts. Section 1.3 finally gives the outline of the remaining parts of the report.

1.1 Background

We expect that most of the readers of this report have read the KobrA book and want to look at the case study described there in more detail. That is, most of the readers know about the KobrA method and also know some of the diagrams presented in this report. Therefore, we ommited any process descriptions and artifact definitions.

In this section, the background of the KobrA method from one of the integrated IESE compentencies' perspective, namely product line engineering, is given. First, PuLSE is introduced as the framework for product line engineering developed at IESE. Then, KobrA is motivated as an instance of the PuLSE framework. Finally, we define the domain selected as subject of interest for the case study at a general level.

1.1.1 PuLSE

Domain engineering[1] has been expected to improve the efficiency of software development because of the notion of economics of scope. Focussing on an

1 see http://www.iese.fhg.de/Domain Engineering for general bibliographic information on domain engineering

area, or domain, where applications significantly overlap enables leveraging the similarities through reuse. Building a reusable infrastructure once for the domain allows multiple applications to be built more efficiently than building them in isolation.

However, domain engineering relies on the notion of an application domain to scope the reusable infrastructure. An application domain spans all possible applications in that domain. Domains have proved difficult to scope and engineer from an enterprise stand point because a domain captures many extraneous elements that are of no interest to an enterprise. Hence, the domain view provides little economic basis for scoping decisions. Instead, enterprises focus on particular products (existing, under development, and anticipated). This difference in focus is essential for practically supporting the product-driven needs of enterprises. Products span, as well as, integrate multiple application domains, yet most often only cover a fraction of these whole domains.

The mission of the IESE is to transfer innovative technologies to our customers to help them improve their software engineering and organization practices. Within that context, we have attempted to transition domain engineering know-how. Our initial approach was to use documented methods, such as Commonality Analysis [AW99], Feature-oriented Domain Analysis [FODA98], or Synthesis [SPC93]. As we used some of their components, problems immediately surfaced: the methods have been either not flexible enough to meet the needs of various industrial situations, or they have been too vague, not applicable without strong additional interpretation and support.

These problems forced us to find solutions throughout the logical phases of the domain engineering lifecycle. Slowly, these solutions together evolved towards an integrated approach of its own: PuLSE™ (Product Line Software Engineering)[1] [BFK+99]. PuLSE is the result of a typical bottom-up effort: the methodology captures and leverages the lessons learned from our technology transfer activities with our industrial customers.Therefore, PuLSE is a flexible method that can be customized to support various enterprise situations.

The lifecycle of a software product line in PuLSE is split into the following phases[2]: initialization, product line infrastructure construction, usage, and evolution. PuLSE provides technical components for the different deployment phases that contain the technical know how needed to operationalize the product line development. The technical components are customizable to the respective context. Customization of PuLSE to the context where it will be applied

1 PuLSE is a registered trademark of the Fraunhofer IESE.
2 see http://www.iese.fhg.de/PuLSE for more information on PuLSE

ensures that the process and products are appropriate. In the initialization phase, the other phases and the technical components are tailored. Through this tailoring of the technical components, customized versions of the construction, usage, and evolution phases of PuLSE are created.

The principle dimensions of customization are the nature of the application domain, the organizational context, reuse aims and practices, as well as the project structure and available resources.

1.1.2 KobrA

PuLSE has been applied successfully in various different contexts for different purposes. Among other things it has proven helpful for introducing sound documentation and development techniques into existing development practices. However, in circumstances where there were no pre-existing processes or well-defined products, the introduction of PuLSE turned out to be problematic. In such cases, the "customization" of PuLSE was actually more concerned with the introduction of basic software engineering processes than with the adaptation of the product line ideas to existing processes. Especially in immature environments, the effort for this process definition can be considerable and even prohibitive.

From the perspective of the PuLSE method, therefore, there is much to be gained by the definition of a "ready-to-use" customization of the method that already contains the required software development processes that may be missing in immature organizations. The result of this effort is the KobrA method [ABB+01][1].

The KobrA method represents a synthesis of several advanced software engineering technologies. Besides product line development, it includes component-based development, frameworks, architecture-centric inspections, quality modeling and process modeling. These have been integrated in KobrA with the basic goal of providing a systematic approach to the development of high-quality, component-based application frameworks. Numerous methods claim to support component based development, but these invariably tend to be rather vague and unprescriptive in nature. They define a lot of possibilities, but provide little if any help in resolving the resulting choices between them. KobrA, in contrast, aims to be as concrete and prescriptive as possible.

1 see http://www.iese.fhg.de/KobrA for more information on the KobrA method and related activities

From a product line perspective KobrA represents an object-oriented customization of the PuLSE method. The infrastructure construction phase of PuLSE corresponds to KobrA's framework engineering activity, the infrastructure usage phase of PuLSE corresponds to KobrA's application engineering activity, and the product line evolution phase of PuLSE corresponds to the maintenance of the frameworks and applications.

The purpose of the framework engineering activity is to create, and later maintain, a generic framework that embodies all product variants in a family, including information about their common and disjoint features. The purpose of the application engineering activity is to instantiate this framework to create particular variants in the product family, each tailored to meet the specific needs of different customers, and later to maintain these concrete variants. A given framework can therefore be instantiated multiple times to yield multiple applications.

1.1.3 Library System Case Study

For our case study it was essential to find a suitable domain with a sufficient spectrum of variability, which is easily understandable. As a customer-oriented example the domain of library systems seems to be a good choice, because nearly everyone is familiar with some features as a user of a modern city library for private purposes or has used a university library during his or her study.

However, beside the above mentioned two types of library, libraries can be classified into far more different library types - each with specific mission, target groups, and tasks. The library types common in Germany can be classified into the following main categories [Hac92]:

- National Libraries
- Central Subject Libraries
- Regional Libraries
- Academic Libraries
- Special Libraries

Each category includes a large variety of library types as shown in the table below [PCM96]. This example offers a sufficient spectrum of variability for a small, but yet non-trivial case study. Three of the above mentioned library types (underlined and printed bold in the table) will serve as examples of product line members.

National Libraries	Central Subject Libraries	Regional Libraries	Academic Libraries	Special Libraries
Die Deutsche Bibliothek • Deutsche Bibliothek Frankfurt am Main • Deutsche Bücherei Leipzig • Deutsches Musikarchiv Berlin	Universitätsbibliothek Technische Informationsbibliothek UB/TIB Hannover	Regional Libraries of a German 'Land' (e. g., Sächsische Landesbibliothek Dresden;Badische Landesbibliothek Karlsruhe;Niedersächsische Landesbibliothek Hannover)	**University Libraries**	Company Libraries
Früher: Deutsche Staatsbibliothek Berlin und Staatsbibliothek Preußischer Kulturbesitz Berlin Jetzt:Staatsbibliothek zu Berlin - Preußischer Kulturbesitz	Deutsche Zentralbibliothek Medizin ZB Med. Köln	**City Libraries**	Libraries of specialized higher education institutions (polytechnic)	**Research Libraries**
Bayerische Staatsbibliothek München	Deutsche Zentralbibliothek für Wirtschaftswissenschaften ZBW Kiel		Departmental Libraries	Parlamentary and Administrative Libraries
	Deutsche Zentralbibliothek für Landbauwissenschaften ZBL Bonn		Institute Libraries	Patent Office Libraries
			Seminar Libraries	Hospital Libraries
				Music Libraries
				Military Libraries
				Church Libraries
				Art Libraries

1.2 Objectives

This section ellaborates on the objectives of the case study project and on the reasons for spending effort on an academic case study instead of demonstrating the KobrA method by material taken from one of our transfer projects in industrial contexts.

The case study was performed to serve two major purposes. On the one hand, a case study was needed to play around with new ideas and to experiment with alternative apporaches for realizing diverse concepts in the method. For this purpose, material from an industrial context is not suitable for two reasons. First, researchers are typically no experts in the particular application domain and they also have no access to the experts in the industrial organization. Hence, it is difficult to work with such a material. Second, industry matrial is typically large and thus it costs much effort to realize a conceptual change consistenly.

On the other hand, the case study was planned as an example that illustrates all aspects and features of the KobrA method. Also for this purpose, material from an industrial context is not suitable for several reasons. First, most parts of industry material is concerned with details of an application domain that is not understood by anybody. Second, such a material is not fully owned by the researchers and thus they are usually not permitted to publish it completey and in all details. Third, industry material is large and thus many aspects of it do not concern method features but simply add complexitiy to the example.

For the given reasons, we decided to do an academic case study that is, on the one hand, small enough to enable our experimental research approach. That is, the effort needed to propagate changes to all models and thus to keep the entire case study consistent was realistic. On the other hand, the academic case study was large and complex enough to illustrate all features and concepts of the KobrA method. Especially the illustration of product line variabilities, their complexity, and the need for special mechanisms for managing them require an appropriate level of complexity.

As described in the previous section, we selected the domain of library and information systems for our case study. This report documents the parts of the case study that complement and complete the examples given in the KobrA book to provide a more complete picture of the KobrA method for interested readers.

1.3 Outline

This report is organized as follows. Part I presents the library system product line. It consists of the scope (chapter 2), the generic context realization (chapter 3), as well as the generic Komponents LibrarySystem (chapter 4), LoanManager (chapter 5), and ReservationManager (chapter 6).

Part II presents a product line member of the library systems product line, the basic library system. It consists of the specific Komponents LibrarySystem (chapter 7) and LoanManager (chapter 8).

Part I Framework for Library Systems

2 Framework Scope

In this chapter, the domain of library and information systems is anaylzed to determine the scope of the case study. In general, the domain of library systems encompasses systems that help librarians and information professionals to perform their work. Library systems provide support for different aspects of library work. This includes assistance in customer interaction, stock management, and accounting. The scope of the domain of library systems is characterized by the common and variable aspects of the different systems in the library systems product line.

We defined the scope of the product line on the basis of three types of library systems: a city library, a university library, and a research library (i.e. the IESE inhouse research library). We selected these three systems as examples, because all three library types are represented in Kaiserslautern. The three product line members are generally characterized in Section 2.1. Section 2.2 defines the features of the domain. Section 2.3 provides an overview of the features and sets them in realtion to the three product line members in form of a product map. Section 2.4 identifies the part of the scope that is used for the book example by presenting a reduced product map.

2.1 Product Line Members

This section characterizes the product line members which were used as basis for defining the scope of the case study. The three types of library systems, namely a city library, a university library, and a research library, are described in the following subsections.

2.1.1 IESE Research Library

The IESE research library is a specialized, scientific library for the researchers, students, and employees working at the Fraunhofer Institute for Experimental Software Engineering (IESE). It is also responsible for different information gathering, documentation, training, and consulting tasks. The purpose of the IESE research library is to supply people at IESE with specialized information based on their specific needs for research and project work.

2.1.2 University Library

A university library is a scientfic library. Its purpose is to provide university members (i.e., students, professors, and other scientific staff) comprehensively with scientific literature. The research areas and the curriculum subjects of the university are covered by the library. In most cases, university libraries are also part of a supra-regional inter-library lending system.

2.1.3 City Library

A city library is a public library. Its purpose is to provide all citizens of a city and the surrounding areas with literature and media for personal education and training, for day-to-day management, and as a basis for forming their own opinions. Its stock includes fiction and non-fiction, poetry, juvenile literature, and periodicals. Further purposes of a city library are to teach competence in the use of media and to promote reading.

2.2 Features

The following is a list of features or services for the library system product line. Those features can be either shared by different systems in the product line or they can be specific. The identification of the features that a specific system provides is done in the product map (cf. Section 2.3).

2.2.1 Catalog

Catalog
The system indexes and stores each library item in a Catalog. Depending on the item type, different data is entered. This data includes bibliographic description and subject indexing. The Catalog provides search, report, and print facilities.

Monograph Catalog
The Catalog provides support for indexing and storing monographs.

Periodical Catalog
The Catalog provides support for indexing and storing periodicals and newspapers.

Issue Catalog
The Catalog provides support for indexing and storing single issues of a journal.

CoP Catalog
The Catalog provides support for indexing and storing component parts (CoPs, e.g., single periodical articles, conference papers, or book chapters).

2.2.2 Item Acquisition

Acquisistion on Approval
Items that are ordered on approval can be entered in the Catalog temporarily. After a given period, these items are either removed from the Catalog or entered permanently.

Item Suggestion
The system supports the process of managing items suggested by library users.

Firm Order
The system supports the process of ordering an item from a supplier.

Order Claim
The system monitors undelivered orders. In case ordered items are not delivered on time by the supplier, a notice of claim is sent to the supplier.

Order Cancellation
The system supports the process of order cancellation.

2.2.3 Periodical Kardex

Kardex
The system provides a kardex for monitoring regular accession and for registering the date of receipt of each issue, depending on the publication frequency of a subscribed periodical.

Trial Subscription
The system supports the handling of trial subscriptions.

Subscription
The system supports subscriptions of periodicals.

Subscription Claim
The system monitors subscribed periodicals using the kardex. In case a subscribed periodical is not delivered on time by the supplier, a notice of claim is sent to the supplier.

Subscription Binding
The system handles periodicals that are bound to volumes when a certain volume is complete.

Routing
The system provides support for the routing (or circulation) of single issues of subscribed periodicals.

2.2.4 Data Management

User Management
The system processes standardized data about the users of the library. This includes support for user registration, account reporting, etc.

Supplier Management
The system processes standardized data about the suppliers of the library.

Publisher Management
The system processes standardized data about publishers of items in the library stock.

Classification Management
The system processes standardized data about the classification used in the library.

Keywords Management
The system processes standardized data about subject headings used to characterize the contents of items in the library.

Descriptor Management
The system processes standardized data about descriptors (i.e., subject headings taken from a controlled vocabulary) used to characterize the contents of items in the library.

2.2.5 Data Exchange

Data exchange covers all ways of communication between the library system and systems that either act as data provider or as data consumer.

Data Consumer
The system can import data provided by another system. For example, an interface to a publication database is available, which provides means to handle all necessary steps involved in storing inhouse items (from registration to putting in library stock).

Data Provider
The system can actively export data to other systems. For example, newly entered data of a specific kind are exported to an external Catalog.

Union Catalog Interface
An interface is provided to a union Catalog that is the result of the co-operative cataloguing performed by a network of libraries. The system then provides Catalog information for download.

Z39.50 Interface
A standardized union Catalog interface.

2.2.6 Loan Management

Loan
The system supports the librarian in checking out an item from the library stock and lending it to a library user for a determined period of time.

Return
The system supports the librarian in checking an item into the library stock that has been returned by a library user.

Renewal
The loan period for a loaned item is renewed.

Reservation
If an item a user wants to loan is already lent to someone else, the user can reserve it. He or she is then informed when the item is available for loan again.

Return Claim
The system monitors loaned items. If the return date of a loaned item is exceeded, a notice of claim is sent to the library user.

2.2.7 Charges

Loan Fee
The system supports the librarian in collecting fees or charges from library users for borrowing items from the library.

Reservation Fee
The system supports the librarian in collecting fees or charges from library users for reserving an item.

Overdue Fee
The system supports the librarian in collecting fees or charges from library users for exceeding the loan period.

Loss Fee
The system supports the librarian in collecting fees or charges from library users for lost items.

2.2.8 Reports and Profiles

Minimal Report Format
The system provides information on items in a minimal report format.

Maximal Report Format
The system provides information on items in a maximal report format.

DIN 1505 Report Format
The system provides information on items in this standardized report format.

Accession Profiling
The system provides library users with the accession list.

Periodical Profiling
The system provides library users with information on the periodicals collection.

2.2.9 OPAC (Online Public Access Catalog)

OPAC
The system enables library users to access their catalog online.

Simple OPAC Search
The OPAC provides simple search facilities.

Advanced OPAC Search
The OPAC provides advanced search facilities.

Expert OPAC Search
The OPAC provides expert search facilities.

OPAC Index
The OPAC provides indices.

OPAC Reference Linking
The OPAC provides reference linking to identify related work.

OPAC Reservation
The OPAC provides facilities to reserve items that are currently on loan online.

OPAC Renewal
The OPAC provides facilities to renew the loan period for an item online.

OPAC Profiling
The OPAC allows library users to define search profiles.

OPAC Borrowers' File
The OPAC allows library users to view their own user account ("borrowers' file") (i.e., loaned items, return dates, and reservations).

2.3 Product Map

A product map displays the relationships between the features identified above and the members of a product line in a two-dimensional matrix. The gray-shaded features are variable among the three products.

		City Library	University Library	IESE Research Library
Catalog	Catalog	X	X	X
	Monograph Catalog	X	X	X
	Periodical Catalog	X	X	X
	Issue Catalog			X
	Article Catalog			X
Item Acquisition	Acquisition on Approval		X	
	Item Suggestion*	X	X	X
	Firm Order	X	X	
	Order Claim	X	X	
	Order Cancellation	X	X	
Periodical Kardex	Kardex		X	X
	Trial Subscription		X	
	Subscription		X	X
	Subscription Claim		X	X
	Subscription Binding		X	
	Routing			X

		City Library	University Library	IESE Research Library
Data Management	User Management*	X	X	X
	Supplier Management	X	X	X
	Publisher Management	X	X	X
	Classification Management*	X	X	X
	Keyword Management		X	X
	Descriptor Management			
Data Exchange	Internal Database Interface			X
	External Database Interface			X
	Union Catalog Interface		X	
	Z39.50 Interface		X	
Loan Management	Loan	X	X	X
	Return	X	X	X
	Renewal	X	X	X
	Reservation	X	X	X
	Return Claim*	X	X	X
Charges	Loan Fee	via annual fee		
	Reservation Fee	X		
	Overdue Fee	X	X	
	Loss Fee	X	X	X
Reports/Profiles	Minimal Report Form		X	X
	Maximal Report Form		X	X
	DIN 1505 Report Format			X
	Accession Profiling	X	X	X
	Periodical Profiling	X	X	X

		City Library	University Library	IESE Research Library
OPAC	OPAC	X	X	X
	Simple OPAC Search	X	X	X
	Extended OPAC Search		X	X
	Expert OPAC Search		X	X
	OPAC Index			X
	OPAC Reference Linking			X
	OPAC Reservation			X
	OPAC Renewal			
	OPAC Profiling			X
	OPAC Borrowers' File			X

* Variabilities at a more detailed level are expected.

2.4 Reduced Product Map

The product map shown in the previous section was the result of relating the identified features to the three systems in the product line. In order to be able to present the case study as running example in the KobrA book, the scope had to be reduced. The result of this feature reduction is shown in the product map below, which is the basis for the remainder of this report. f

		K'Town City Library	University Library	IESE Library
Customer Management	Registration	X	X	X
	Unregistration	X	X	X
	Registration Change	X	X	X
Loan Management	Loan	X	X	X
	Return	X	X	X
	Report Loss	X	X	X
	Item Reservation		X	X
	Item Suggestion		X	X
Stock Management	Overdue Control	X	X	X
	Inventorying	X	X	X
	Statistics		X	X
	Classification Management	X	X	X
	Keyword Management		X	X
	Descriptor Management		X	

		K'Town City Library	University Library	IESE Library
Item Management	Item Acquisition	X	X	X
	Item Registration	X	X	X
	Item Removal	X	X	X
Periodical Management	Subscription Acquisition		X	X
	Periodical Registration		X	X
	Periodical Monitoring		X	X
	Periodical Contents Registration		X	X
	Periodical Removal		X	X
	Periodical Unsubscribing		X	X
Data Exchange	Data Import		X	X
	Data Export		X	X
Account-ing	Billing	X		
	OPAC		X	X

3 Context Realization

Context realization activities include a complete and detailed analysis of the organization behind the library visible to its customers. Unfortunately, such an analysis could not be done for the libraries that are part of this case studies. The reason was the significant effort required from both sides, the external analyzer and the people in the library organization, that is simply too big to be useful in the context of a case study only. Therefore, the context realization activities were done more abstract and from an external point-of-view. That is, organization issues impacting the implementation of business processes related to the library system are not modeled in great detail. As a consequence, the borders between business processes independent of a library system to support them, their refinement into use cases, and finally into system services is not very sharp. In short, the models presented are only examples to illustrate KobrA's context realization models and their inter-relationships. The models are a means to support people interested in the KobrA method to understand the role of a context realization. They are not meant to provide insights into real world library organizations.

3.1 Enterprise Model

3.1.1 Enterprise Concept Diagram

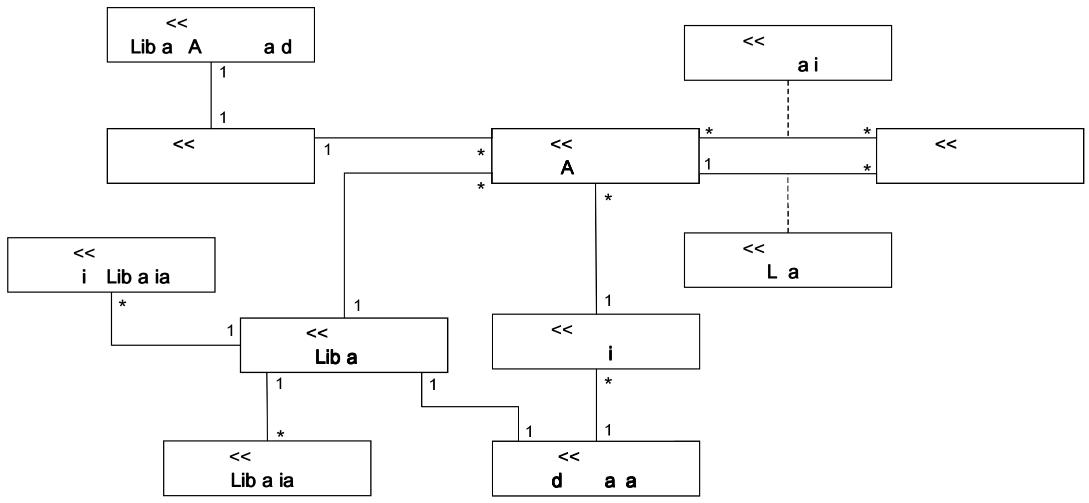


Figure 1: Enterprise Concept Diagram

3.1.2 Enterprise Process Diagram

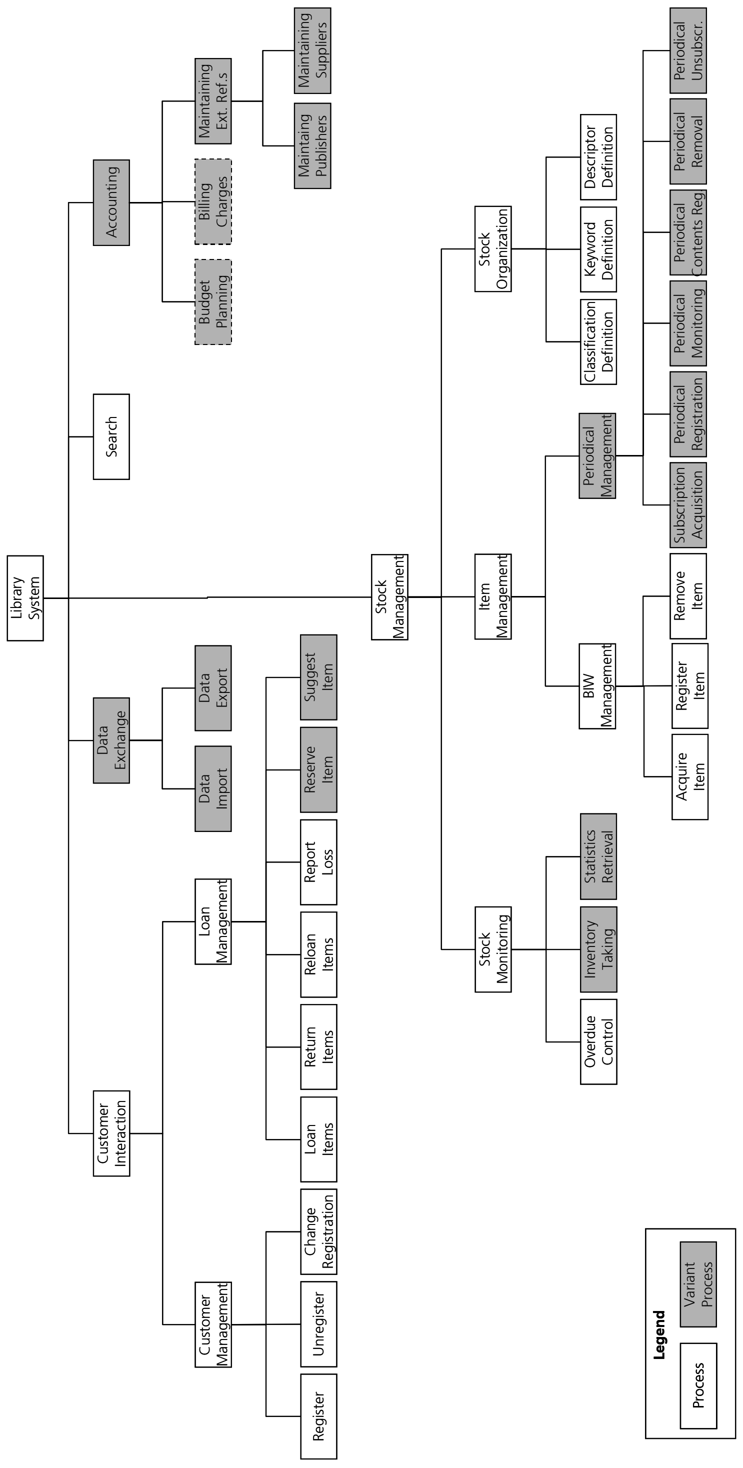


Figure 2: Enterprise Process Diagram

3.2 Structural Model

3.2.1 Context Realization Class Diagram

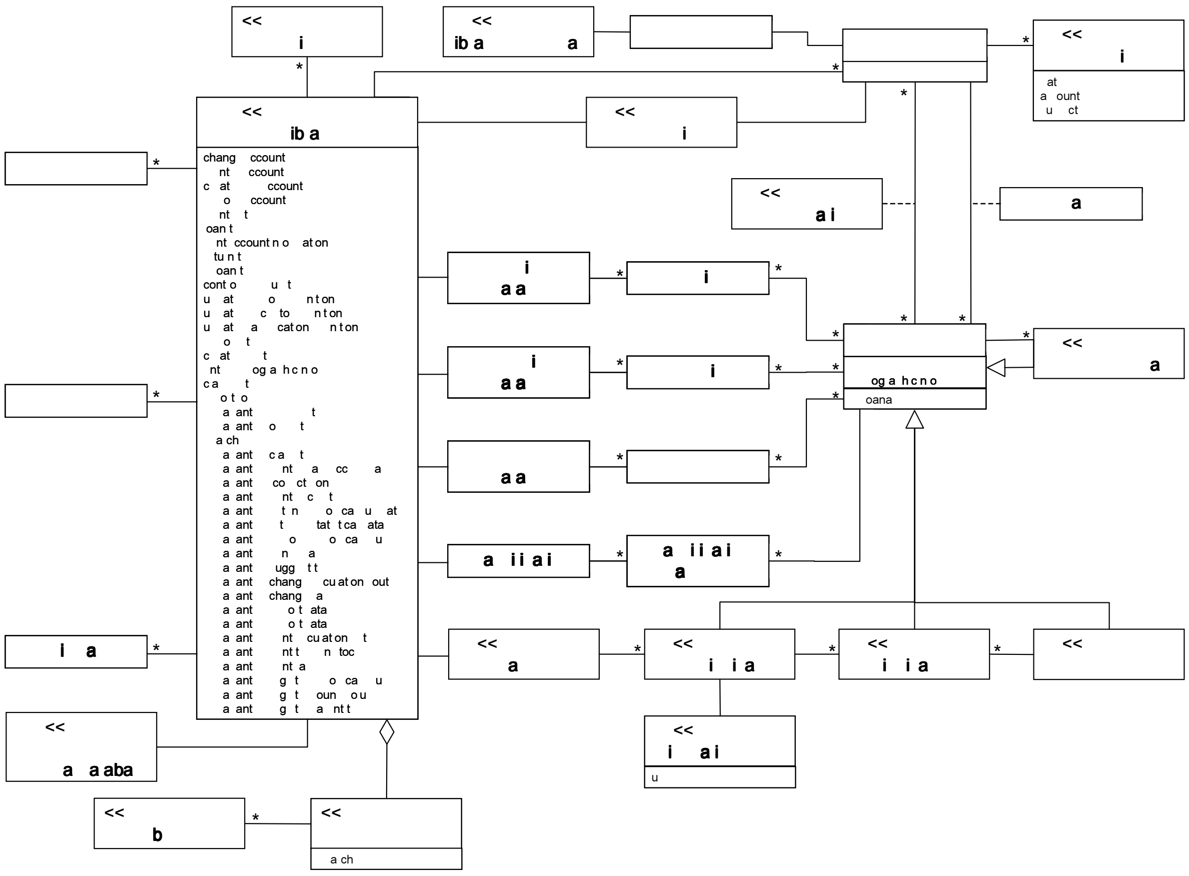


Figure 3: Context Realization Class Diagram

3.2.2 Context Realization Object Diagram

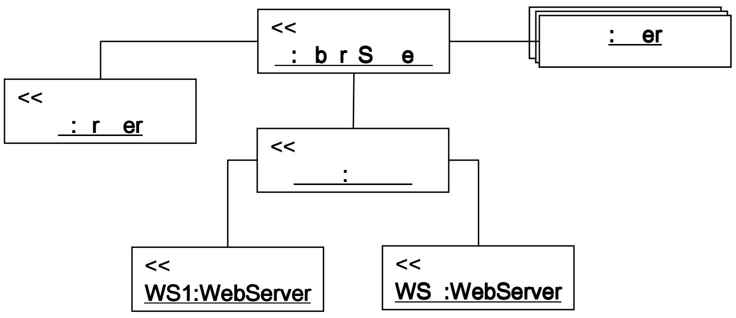


Figure 4: Context Realization Class Diagram

3.3 Activity Model

The activity model plays an important role in projects where the target organization(s) are modeled in detail. As mentioned above, this has not been done in this case study, therefore, the activity model does not clearly work out the difference between business processes, use cases, and system services. The processes presented here are performed mainly by a single role, the service librarian, who directly interacts with the users of the library.

3.3.1 Activity Diagrams

In this section, the activities related to loan and reservation functionality are presented.

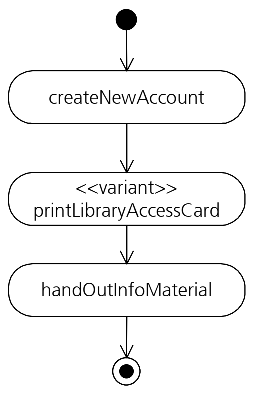


Figure 5: Activity Diagram Register()

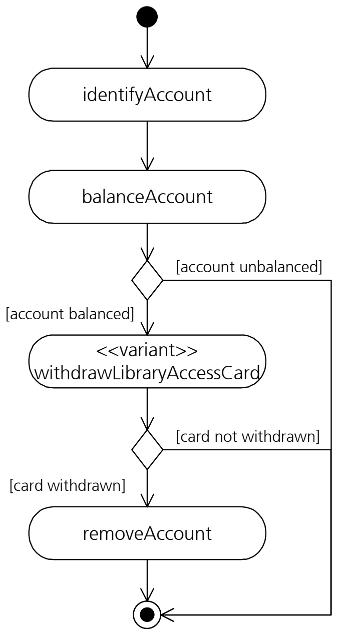


Figure 6: Activity Diagram Unregister()

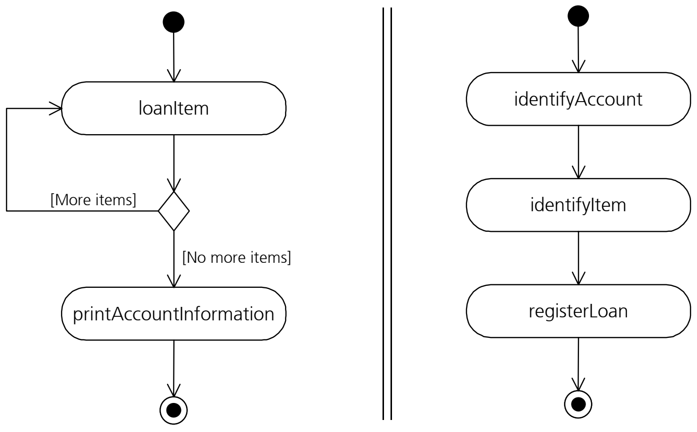


Figure 7: Activity Diagram LoanItems() and a refinement of loanItem()

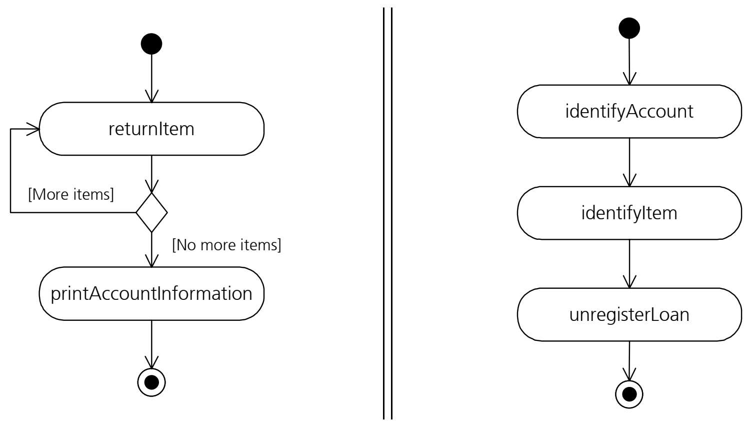


Figure 8: Activity Diagram returnItems() and a refinement of returnItem()

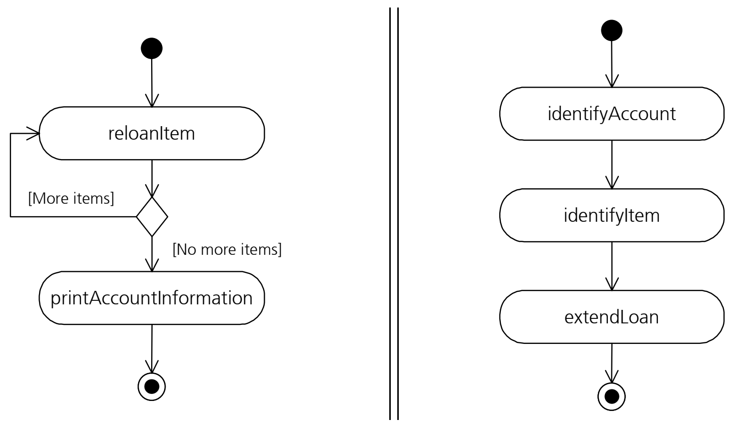


Figure 9: Activity Diagram reloanItems() and a refinement of reloanItem()

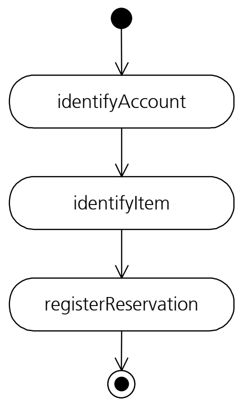


Figure 10: Activity Diagram reserveItem()

3.3.2 Use Case Model

The context realization activities begin with the business processes shown above in form of a business process hierarchy (see Figure 2), and then refines these processes as shown above using activitiy diagrams. The activities in the business process refinements can also be arranged into a hierarchy, which can be integrated with the business process hierarchy. For example, Figure 11 refines the business process in the area customer management and loan management and exactly contains the activities used in the activity models above.

In this hierarchy, some layers can be identified as the activities usually understood as use cases (cmp. with grayed activities). Some of the use cases are not supported by the library system (printLibraryAccessCard, withdrawLibraryAccess-Card, and handOutInfoMaterial). Others are too fine-granular to be seen as a use case, such as registerLoan, extendLoan, or balanceAccount. The use cases identified are presented in the use case diagram shown in Figure 12.

3.4 Interaction Model

3.4.1 Sequence Diagrams

The interaction model ultimately assigns behavior to objects (representing data). That is, data and behavior are integrated with each other. An object-oriented view on the system is created. Here, some examples of sequence diagrams are given related to the activity diagrams presented above. (Note that activity dia-

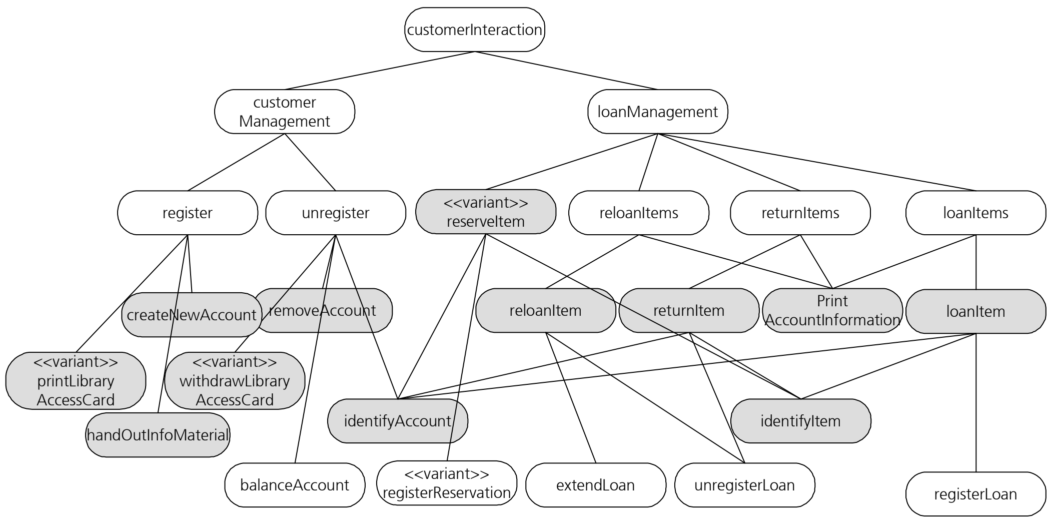


Figure 11: Activity hierarchy

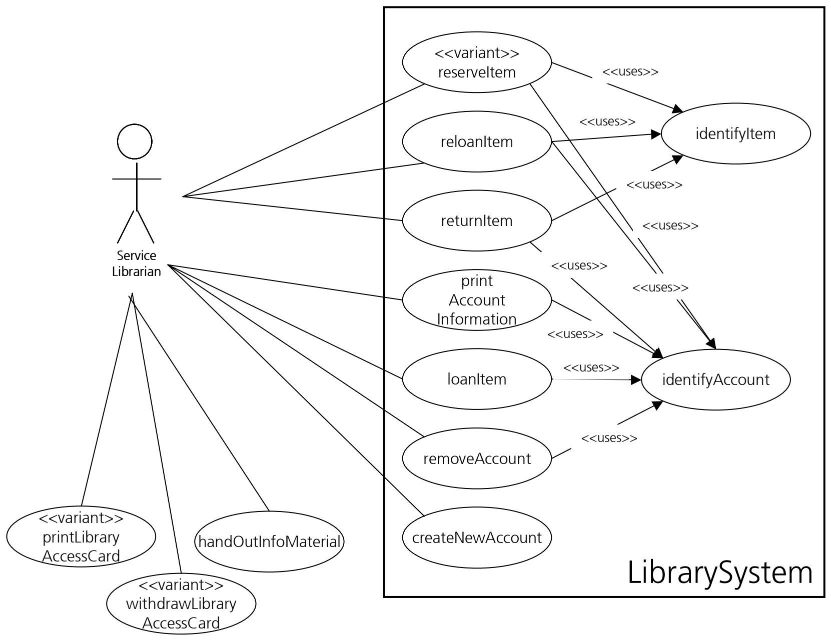


Figure 12: Use case diagram for the actor ServiceLibrarian

grams with swim-lanes are, in general, equivalent to sequence or collaboration diagrams.

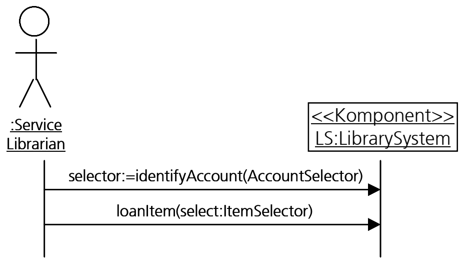

:Service Librarian
<<Komponent>> LS:LibrarySystem
selector:=identifyAccount(AccountSelector)
loanItem(select:ItemSelector)

Figure 13: Sequence diagram for loanItem()

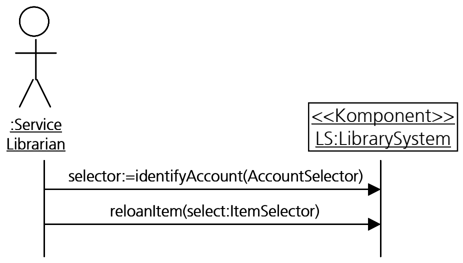

:Service Librarian
<<Komponent>> LS:LibrarySystem
selector:=identifyAccount(AccountSelector)
reloanItem(select:ItemSelector)

Figure 14: Sequence diagram for reserveItem()

3.5 Decision Model

ID	Variation	Resolution	Effect
CR-1	Reservation	yes (default)	yes: CR1.1, CR2.1, CR5.1
		no	no: CR1.1, CR2.1, CR5.1
CR-2	External Database	yes	yes: CR1.2, CR2.2, ...
		no (default)	no: LS13.2, LS14.1, ...
CR-3	Suggestion	yes (default)	yes: CR1.3, CR2.3, ...
		no	no: CR1.3, CR2.3, ...
CR-4	LibraryAccessCard	yes	yes: CR2.4, CR3.1, CR4.1, CR5.2, ...
		no	no: CR2.4, CR3.1, CR4.1, CR5.2, ...
...	...	...	...

Table 1: Integrated Decision Model for the LibrarySystem Context Realization

ID	Variation	Resolution	Effect
There is no variability in the enterprise concept model presented in this report. This kind of diagram will contain variability when the analysis of the target organizations is performed in detail. To which extend it is possible or useful to integrate different organizations in an integrated model depends on the domain and the variety of customer organizations.			

Table 2: Decision Model for Enterprise Concept Model (Figure 1)

ID	Variation	Resolution	Effect
CR1.1	Reservation	yes (default)	—
		no	remove process reserveItem
CR1.2	External Database	yes	—
		no (default)	remove process data exchange (and its subprocesses)
CR1.3	Suggestion	yes (default)	—
		no	remove process suggestItem
...	...	...	...

Table 3: Decision Model for the enterprise process diagram (Figure 2)

ID	Variation	Resolution	Effect
CR2.1	Reservation	yes (default)	—
		no	remove method LibrarySystem.reserveItem() remove association class Reservation
CR2.2	External Database	yes	—
		no (default)	remove Komponent ExternalDatabase remove method LibrarySystem.importData remove method LibrarySystem.exportData
CR2.3	Suggestion	yes (default)	—
		no	remove class Suggestion remove method LibrarySystem.suggestItem()

Table 4: Decision Model for Context Realization Class Diagram (Figure 3)

ID	Variation	Resolution	Effect
CR2.4	LibraryAccessCard	yes	—
		no	remove class LibraryAccessCard
…	…	…	…

Table 4: Decision Model for Context Realization Class Diagram (Figure 3)

ID	Variation	Resolution	Effect
CR3.1	Library Access Card	yes	—
		no	remove activity printLibraryAccessCard

Table 5: Decision Model for Activity DIagram register() (Figure 5)

ID	Variation	Resolution	Effect
CR4.1	Library Access Card	yes	—
		no	remove activity withdrawLibraryAccessCard

Table 6: Decision Model for Activity DIagram unregister() (Figure 6)

ID	Variation	Resolution	Effect
CR5.1	Reservation	yes (default)	—
		no	remove process reserveItem
CR5.2	LibraryAccessCard	yes	—
		no	remove use case printLibraryAccessCard remove use case withdrawLibraryAccessCard
…	…	…	…

Table 7: Decision Model for the use case diagram (Figure 12)

4 Library System

4.1 Specification

The specification of LibrarySystem is not presented completely but this report focuses on loan and reservation-related functionality in the library domain. Therefore, some of the models are incomplete with respect to the scope definition given in the previous chapter.

4.1.1 Structural Model

4.1.1.1 Class Diagram

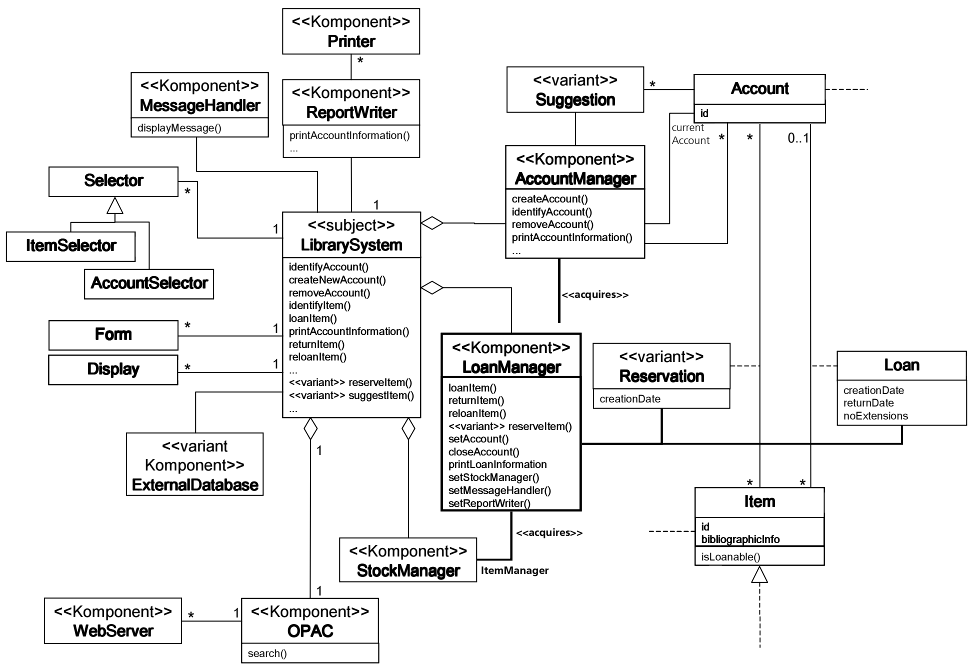


Figure 15: LibrarySystem Specification Class Diagram

4.1.1.2 Supplied and Required Interfaces

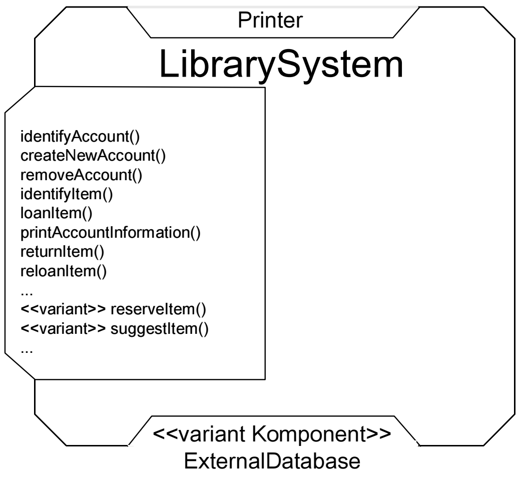


Figure 16: Supplied and Required Interfaces of the LibrarySystem Komponent

4.1.1.3 Object Diagram

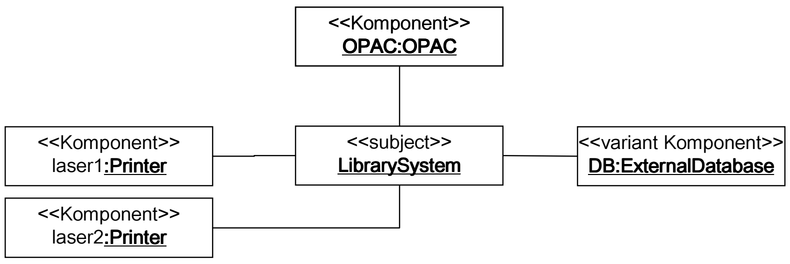


Figure 17: LibrarySystem Specification Object Diagram

4.1.2 Functional Model

4.1.2.1 Operation Specifications

loanItem

Name	loanItem()
Description	The loan of an Item to currentAccount is registered
Receives	selector: ItemSelector
Sends	<variant> Message “Reserved” </variant> Message “Already Loaned”
Rules	An item is loanable if it is not an item that must always stay in the library (e.g., antique books). An item is currently loanable if it is loanable and not loaned <variant> or reserved </variant> by another user.
Changes	new Loan
Assumes	Subject is in the state accountIdentified Selector selects exactly one Item
Result	item selected by selector has been obtained if item is currently loanable a new Loan object, loan, has been created that relates item and currentAccount and has the attribute values - creationDate = today - returnDate = today + <loanPeriod> and - noExtensions = 0 and, loan has been stored. if item is not currently loanable one of the messages has been displayed to the user <variant> - “Reserved” or </variant> - "Already Loaned"

returnItem

Name	returnItem()
Description	Makes an item loanable again <variant> and returns a message if the item is reserved </variant>.
Receives	selector:ItemSelector
Sends	<variant> Message “Reserved” </variant>
Rules	—
Changes	destroy loan

Assumes	subject is in the state accountIdentified selector selects exactly one item item is loaned to currentAccount
Result	item selected by selector has been obtained the loan for item and currentAccount has been destroyed <variant> if item is reserved, the message displayMessage(“item reseverved”) has been sent to MH </variant>

reloanItem

Name	reloanItem()
Description	An item loaned is reloaned to the currentAccount
Receives	selector:ItemSelector
Sends	<variant> Message “Reserved” </variant> Message “Over Extension”
Rules	An item is reloanable if it is loanable and the number of extension is less or equal to <maxExtensions> <variant> and it is not reserved </variant>.
Changes	loan
Assumes	Subject is in the state accountIdentified selector selects exactly one item Item is loaned to currentAccount
Result	item selected by selector has been obtained if item is reloanable the loan containing item has the attrbibute values - returnDate = today + <loanPeriod> - noExtensions = noExtensions+1 if item is not reloanable one of the following messages has been sent to MH <variant> - displayMessage(“Reserved”) or </variant> - displayMessage(“OverExtensions”)

<<variant>> reserveItem

Name	<<variant>> reserveItem()
Description	A reservation fo an item is registered to currentAccount.
Receives	selector:ItemSelector
Sends	Message “Not Reservable”
Rules	An item is reservable if it is loanable.
Changes	new Reservation

Assumes	subject is in the state accountIdentified selector selects exactly one item
Result	item selected by selector has been obtained if item is reservable a new Reservation has been created that relates item and currentAccount and has the attribute value - creationDate = today and reservation has been stored if item is not reservable the message “not reservable” has been sent

printAccountInformation

Name	printAccountInformation()
Description	All information concerning the current account is printed.
Receives	—
Sends	Printer.print(data)
Rules	—
Changes	—
Assumes	subject is in state accountIdentified
Result	The data capturing customer data, current loans <variant>, and reservations </variant> has been obtained, formatted, and sent to Printer.

createNewAccount

Name	createNewAccount()
Description	An account is created for a new customer
Receives	selector:AccountSelector
Sends	—
Rules	—
Changes	new Account
Assumes	selector does not select any existing account
Result	A new account:Account has been created according to the attributes of selector.

identifyAccount

Name	identifyAccount()
Description	An existing account is identified and opened.
Receives	selector:AccountSelector
Sends	—
Rules	—
Changes	currentAccount

Assumes	selector selects exactly one Account
Result	account:Account selected by selector has been obtained Library has been transitioned to state accountIdentified with currentAccount=account

removeAccount

Name	removeAccount()
Description	The currently selected account is closed and removed from the library
Receives	—
Sends	Message “Return all items first”
Rules	—
Changes	destroy loan
Assumes	subject is in the state accountIdentified
Result	If no Loans are related to currentAccount, <variant> first, all existing reservations related to currentAccount are removed, then </variant> currentAccount is destroyed.

4.1.3 Behavioral Model

4.1.3.1 Statechart Tables

In general, the behavioural model of systems of this size (or even bigger) is too complex for being captured in a useful state diagram. But if we hide the process support of the library system and focus only when which operations can be invoked, three main states of the library system can be identified:

- neutral: The system has no state information.
- accountIdentified: A particular account has been identified, which will be supplied to invoked operations (typically services with direct customer interaction with the library work via the customer account).
- itemIdentifed: A particular item has been identifed, which will be supplied to invoked operation (typically services in the background to maintain the library’s stock etc. are item-centric tasks)

The following statechart table lists for each of these three states the operations that can be invoked directly (i.e., not within a process execution) from a user.

No State Information	• identifyAccount • identifyItem • createNewAccount • ...
accountIdentified	• removeAccount • loanItem • printAccountInformation • returnItem • reloanItem • <<variant>> reserveItem • <<variant>> suggestItem • ...
itemIdentified	• reportLoss • ...

4.1.3.2 State Diagram

The statechart table can be translated into the form of a UML statechart diagram. Here, the state diagram illustrates the conceptual states of the library system.

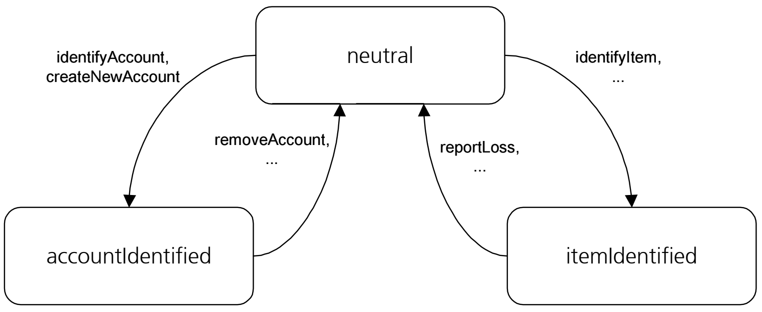


Figure 18:
State Transition Diagram:
The main states of a library system

4.1.4 Decision Model

In the scope definition given in the second chapter of this report, the considered library systems covered various varying concepts. In this report, we focus on variability related to features handling loaning and reserving items in a library. The generic specification of the komponent LibrarySystem varies in the following features:

- Reservation: support for reservations
- External Database: support for data ecxhange with an external database
- Suggestion: support for suggestions

- maxExtensions: the number of extensions a customer can get on a loaned item (integer value)
- loanPeriod: the length of a loan period (time value)

ID	Variation	Resolution	Effect
LS-S1	Reservation	yes (default)	yes: LS1.1, LS2.1, LS4.1, LS5.1, LS6.1, LS7.1, LS8.1, LS11.1, LS12.1, LS-R1
		no	no: LS1.1, LS2.1, LS4.1, LS5.1, LS6.1, LS7.1, LS8.1, LS11.1, LS12.1, LS-R1
LS-S2	External Database	yes	yes: LS1.2, LS2.2, LS3.1, LS12.1, ..., LS-R2
		no (default)	no: LS1.2, LS2.2, LS3.1, LS12.1, ..., LS-R2
LS-S3	Suggestion	yes (default)	yes: LS1.3, LS2.3, LS12.2, ..., LS-R3
		no	no: LS1.3, LS2.3, LS12.2, ..., LS-R3
LS-S4	loanPeriod	value [time]	replace loanPeriod by actual value: LS4.2, LS6.2
LS-S5	maxExtensions	value [int]	replace maxExtensions by actual value: LS6.3

Table 8: Integrated Decision Model for LibrarySystem Specification

ID	Variation	Resolution	Effect
LS1.1	Reservation	yes (default)	—
		no	remove method LibrarySystem.reserveItem() remove association class Reservation
LS1.2	External Database	yes	—
		no (default)	remove Komponent ExternalDatabase remove method LibrarySystem. ...
LS1.3	Suggestion	yes (default)	—
		no	remove class Suggestion remove method LibrarySystem.suggestItem()

Table 9: Decision Model for LibrarySystem Specification Class Diagram(Figure 15)

ID	Variation	Resolution	Effect
LS2.1	Reservation	yes (default)	—
		no	remove method reserveItem()
LS2.2	External Database	yes	—
		no (default)	remove required Interface ExternalDatabase remove method ...
LS2.3	Suggestion	yes (default)	—
		no	remove method suggestItem()

Table 10: Decision Model for LibrarySystem Specification Supplied and Required Interfaces (Figure 16)

ID	Variation	Resolution	Effect
LS3.1	External Database	yes	—
		no (default)	remove Komponent ExternalDatabase

Table 11: Decision Model for LibrarySystem Specification Object Diagram (Figure 17)

ID	Variation	Resolution	Effect
LS4.1	Reservation	yes (default)	—
		no	remove variant tags and content
LS4.2	loanPeriod	value [time]	replace loanPeriod by actual value

Table 12: Decision Model for LibrarySystem.loanItem() operation schema

ID	Variation	Resolution	Effect
LS5.1	Reservation	yes (default)	—
		no	remove variant tags and content

Table 13: Decision Model for LibrarySystem.returnItem() operation schema

ID	Variation	Resolution	Effect
LS6.1	Reservation	yes (default)	—
		no	remove variant tags and content
LS6.2	loanPeriod	value [time]	replace loanPeriod by actual value
LS6.3	maxExtensions	value [int]	replace maxExtensions by actual value

Table 14: Decision Model for LibrarySystem.reloanItem() operation schema

ID	Variation	Resolution	Effect
LS7.1	Reservation	yes (default)	—
		no	exclude operation specification from functional model

Table 15: Decision Model for LibrarySystem.reserveItem() operation schema

ID	Variation	Resolution	Effect
LS8.1	Reservation	yes (default)	—
		no	remove variant tags and content in result clause

Table 16: Decision Model for LibrarySystem.printAccountInformation() operation schema

ID	Variation	Resolution	Effect
LS11.1	Reservation	yes (default)	—
		no	remove variant tags and content in result clause

Table 17: Decision Model for LibrarySystem.removeAccount() operation schema

ID	Variation	Resolution	Effect
LS12.1	Reservation	yes (default)	—
		no	remove method reserveItem()

Table 18: Decision Model for LibrarySystem Statechart Table

ID	Variation	Resolution	Effect
LS12.1	External Database	yes (default)	—
		no	remove method ...
LS12.2	Suggestion	yes (default)	—
		no	remove method suggestItem()

Table 18: Decision Model for LibrarySystem Statechart Table

4.2 Realization

The Realization of LibrarySystem focuses here on services that LibrarySystem assigns to the subkomponent LoanManager. Functionality that is aquired by other subcomponents of LibrarySystem from LoanManager is not taken into account.

4.2.1 Structural Model

4.2.1.1 Class Diagram

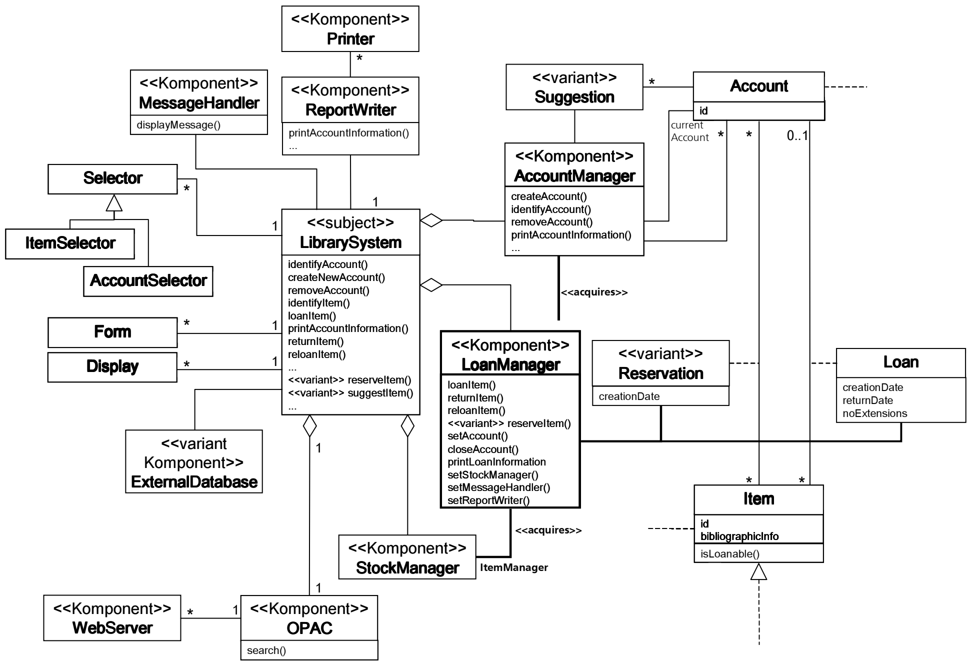
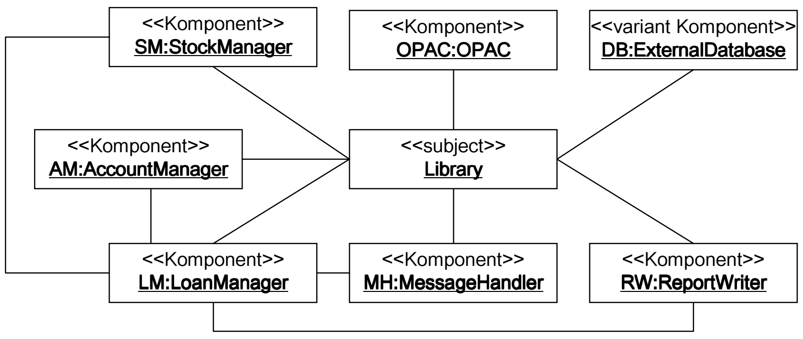


Figure 19: Library Realization Class Diagram

4.2.1.2 Object Diagram



<<Komponent>> SM:StockManager
<<Komponent>> OPAC:OPAC
<<variant Komponent>> DB:ExternalDatabase
<<Komponent>> AM:AccountManager
<<subject>> Library
<<Komponent>> LM:LoanManager
<<Komponent>> MH:MessageHandler
<<Komponent>> RW:ReportWriter

Figure 20: Library Realization Object Diagram

4.2.2 Activity Model

The simplicity of the activities taken into account in this report (i.e., activities related to loaning and reserving items) allowed us to realize them without intermediate refinement steps.

4.2.3 Interaction Model

In general, there are three ways for a subkomponent to be involved in the realization of its parent komponent's services:

- Delegation: a parent komponent does not add anything to the services provided by a subkomponent
- Synchronization: the state of the parent komponent changes in a way that requires a state change of the subkomponent to keep the system consistent.
- Usage: the parent komponent integrates numberous services of subkomponents to realize its (more powerful) services

4.2.3.1 Collaboration Diagrams

In our case of the loan and reservation functionality, the LibrarySystem fully delegates these services to a subkomponent (i.e., LoanManager)

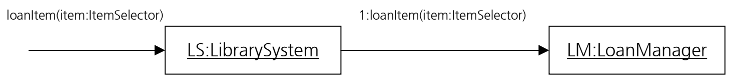


Figure 21: Collaboration Diagram for the loanItem() Operation

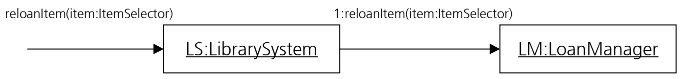

reloanItem(item:ItemSelector)
1:reloanItem(item:ItemSelector)
LS:LibrarySystem
LM:LoanManager

Figure 22: Collaboration Diagram for the reloanItem() Operation

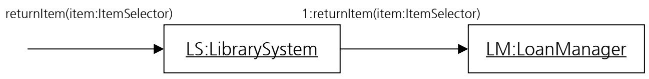

returnItem(item:ItemSelector)
1:returnItem(item:ItemSelector)
LS:LibrarySystem
LM:LoanManager

Figure 23: Collaboration Diagram for the returnItem() Operation

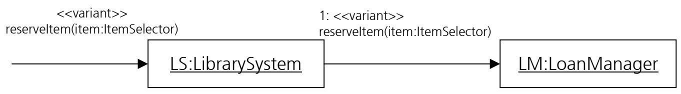

<<variant>> reserveItem(item:ItemSelector)
1: <<variant>> reserveItem(item:ItemSelector)
LS:LibrarySystem
LM:LoanManager

Figure 24: Collaboration Diagram for the reserveItem() Operation

The LoanManager is not a stateless component but requires to specify an account first (currentAccount) which is then subject of subsequent actions. The AccountManager is responsible for Accounts independent of the LonaManager, therefore, some usages of AccountManager by the LibrarySystem requires an explicit state synchronization with the LoanManager komponent.

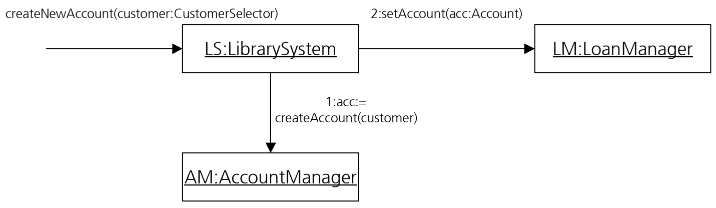

createNewAccount(customer:CustomerSelector)
2:setAccount(acc:Account)
LS:LibrarySystem
LM:LoanManager
1:acc:= createAccount(customer)
AM:AccountManager

Figure 25: Collaboration Diagram for the createNewAccount() Operation

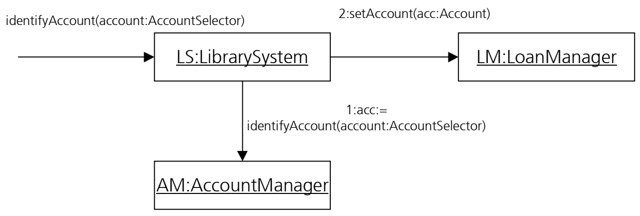

identifyAccount(account:AccountSelector)
2:setAccount(acc:Account)
LS:LibrarySystem
LM:LoanManager
1:acc:= identifyAccount(account:AccountSelector)
AM:AccountManager

Figure 26: Collaboration Diagram for the identifyAccount() Operation

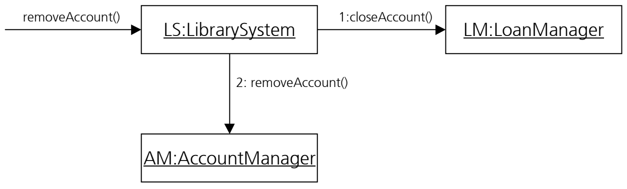


Figure 27: Collaboration Diagram for the removeAccount() Operation

The LibrarySystem allows account information to be printed in a single report. This information is spread over two of its subkomponents: AccountManager and LoanManager. Therefore, to provide the service of printing account information, LibrarySystem must use and coordinate services of its subkomponents.

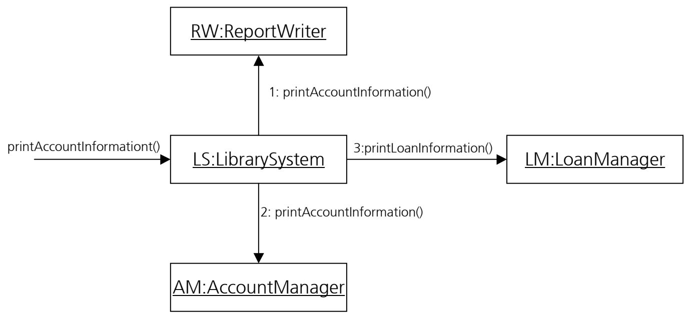


Figure 28: Collaboration Diagram for the printAccountInformation() Operation

4.2.4 Decision Model

ID	Variation	Resolution	Effect
LS-R1	Reservation	yes (default)	yes: LS13.1, LS15.1
		no	no: LS13.1, LS15.1
LS-R2	External Database	yes	yes: LS13.2, LS14.1, ...
		no (default)	no: LS13.2, LS14.1, ...
LS-R3	Suggestion	yes (default)	yes: LS13.3, ...
		no	no: LS13.3, ...

Table 19: Integrated Decision Model for LibrarySystem Realization

ID	Variation	Resolution	Effect
LS13.1	Reservation	yes (default)	—
		no	remove method LibrarySystem.reserveItem() remove association class Reservation
LS13.2	External Database	yes	—
		no (default)	remove Komponent ExternalDatabase remove method LibrarySystem. ...
LS13.3	Suggestion	yes (default)	—
		no	remove class Suggestion remove method LibrarySystem.suggestItem()

Table 20: Decision Model for LibrarySystem Realization Class Diagram(Figure 19)

ID	Variation	Resolution	Effect
LS14.1	External Database	yes	—
		no (default)	remove Komponent ExternalDatabase

Table 21: Decision Model for LibrarySystem Specification Object DIagram (Figure 20)

ID	Variation	Resolution	Effect
LS15.1	Reservation	yes (default)	—
		no	exclude collaboration diagram from interaction model

Table 22: Decision Model for LibrarySystem.reserveItem() Collaboration Diagram (Figure 24)

5 Loan Manager

5.1 Specification

5.1.1 Structural Model

5.1.1.1 Class Diagram

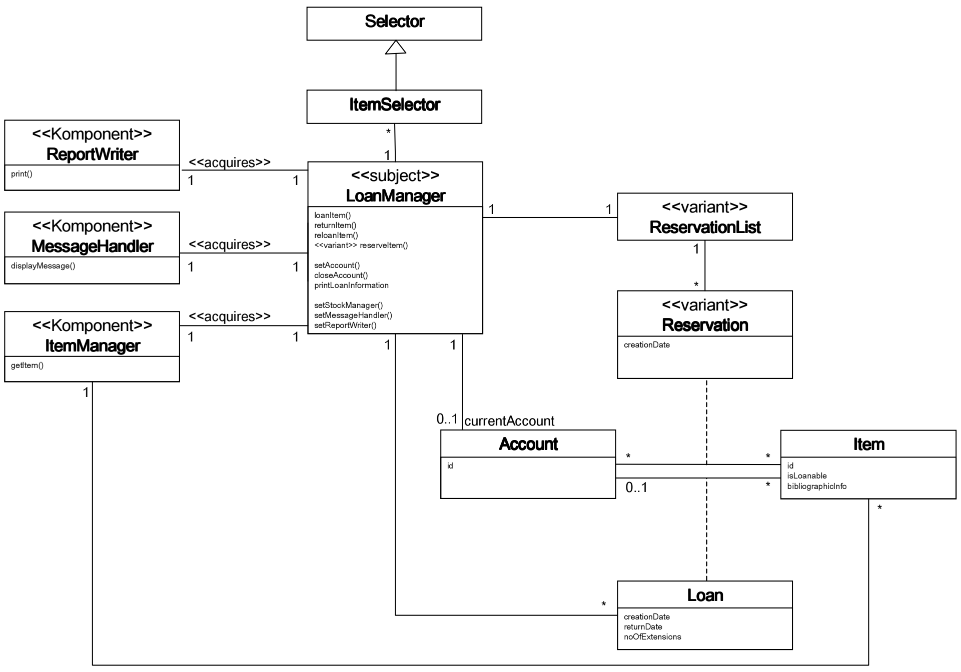


Figure 29: LoanManager Specification Class Diagram

5.1.1.2 Supplied and Required Interfaces

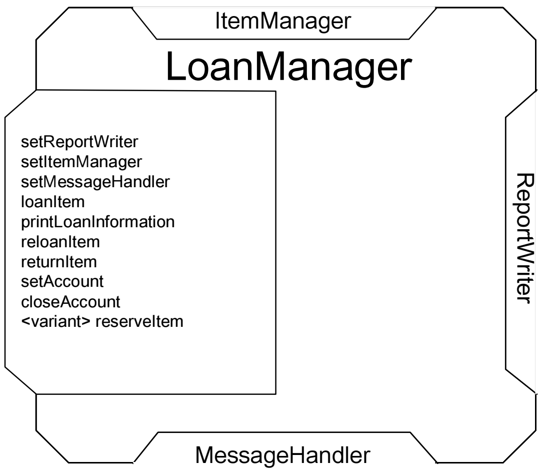


Figure 30: Supplied and Required Interfaces of the LoanManager Komponent

5.1.1.3 Object Diagram

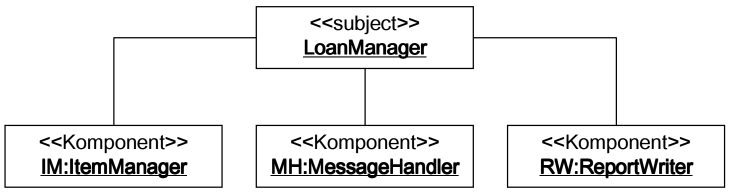


Figure 31: LoanManager Specification Object Diagram

5.1.2 Functional Model

5.1.2.1 Operation Specifications

setAccount

Name	setAccount()
Description	Receives a reference to an Account and stores it as currentAccount
Receives	currentAccount:Account
Sends	—
Rules	—
Changes	—
Assumes	—
Result	currentAccount has been stored and subject is in state accountIdentified

closeAccount

Name	closeAccount()
Description	The current account is closed.
Receives	—
Sends	—
Rules	—
Changes	—
Assumes	—
Result	currentAccount is empty and subject is in state noAccountIdentified

loanItem

Name	loanItem()
Description	The loan of an Item to currentAccount is registered
Receives	selector: ItemSelector
Sends	MH.displayMessage() Item item = IM.getItem(filter)
Rules	An item is loanable if it is not when it is not not an item that must always stay in the library (e.g., antique books). An item is currently loanable if it is loanable and not loaned <variant> or reserved by another user </variant>.
Changes	new Loan
Assumes	Subject is in the state accountIdentified Selector selects exactly one Item
Result	item selected by selector has been obtained from the ItemManager IM by sending the message getItem(filter) if item is currently loanable a new Loan object, loan, has been created that relates item and account, and has the attribute values - creationDate = today - returnDate = today + <loanPeriod> and - noExtensions = 0 and, loan has been stored. if item is not currently loanable one of the messages has been sent to MH <variant> - displayMessage(“Reserved”) or <variant> - displayMessage("Already Loaned")

returnItem

Name	returnItem()
Description	Makes an item loanable again <variant> and returns a message if the item is reserved </variant>.
Receives	selector:ItemSelector
Sends	MH.displayMessage() item Item = IM.getItem(selector)
Rules	—
Changes	destroy loan
Assumes	subject is in the state accountIdentified item is loaned to currentAccount
Result	item selected by selector has been obtained from the ItemManager IM by sending the message getItem(selector) the loan for item and currentAccount has been destroyed <variant> if item is reserved, the message displayMessage(“item reseverved”) has been sent to MH </variant>

reloanItem

Name	reloanItem()
Description	An item loaned is reloaned to the currentAccount
Receives	selector:ItemSelector
Sends	MH.displayMessage() Item item = IM.getItem(selector)
Rules	An item is reloanable if it is loanable and the number of extension is less or equal to <maxExtensions> <variant> and it is not reserved </variant>.
Changes	loan
Assumes	Subject is in the state accountIdentified Item is loaned to currentAccount
Result	item selected by selector has been obtained from the ItemManager IM by sending the message getItem(selector) if item is reloanable the loan containing item has the attrbibute values - returnDate = today + <loanPeriod> - noExtensions = noExtensions+1 if item is not reloanable one of the following messages has been sent to MH <variant> - displayMessage(“Reserved”) or <variant> - displayMessage(“OverExtensions”)

reserveItem

Name	<variant> reserveItem()
Description	A reservation fo an item is registered to currentAccount.
Receives	selector:ItemSelector
Sends	MH.displayMessage() item Item = IM.getItem(selector)
Rules	An item is reservable iff it is loanable.
Changes	new Reservation
Assumes	subject is in the state accountIdentified
Result	item selected by selector has been obtained from the ItemManager IM by sending the message getItem(selector) if item is reservable a new Reservation has been created that relates item and currentAccount and has the attribute value - creationDate = today and reservation has been stored if item is not reservable the message displayMessage(“not reservable”) has been sent to MH

printLoanInformation

Name	printLoanInformation()
Description	All information concerning loans of currentAccount is printed.
Receives	—
Sends	RW.printLoanInfo(currentAccount)
Rules	—
Changes	—
Assumes	subject is in state accountIdentified
Result	A message printLoanInfo(currentAccount) has been sent to RW.

setItemManager

Name	setItemManager()
Description	Receives a reference to an ItemManager komponent and stores it.
Receives	IM:ItemManager
Sends	
Rules	
Changes	
Assumes	
Result	IM has been stored

setMessageHandler

Name	setMessageHandler()
Description	Receives a reference to a MessageHandler komponent and stores it.
Receives	MH:MessageHandler
Sends	
Rules	
Changes	
Assumes	
Result	MH has been stored

setReportWriter

Name	setReportWriter()
Description	Receives a reference to a reportWriter komponent and stores it.
Receives	RW:ReportWriter
Sends	
Rules	
Changes	
Assumes	
Result	RW has been stored

5.1.3 Behavioral Model

5.1.3.1 Statechart Diagram

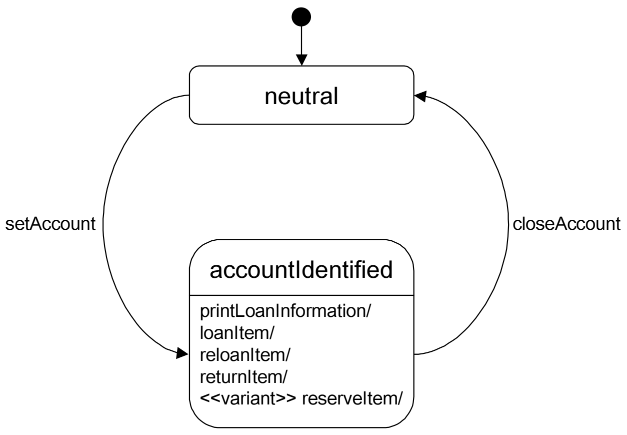


Figure 32: LoanManager Statechart Diagram

5.1.3.2 Statechart Tables

Source State	UML Transition String	Target State
neutral	setAccount()	accountIdentified
accountIdentified	closeAccount()	neutral

accountIdentified	printLoanInformation()
	loanItem()
	reloanItem()
	returnItem()
	<<variant>>reserveItem()
		accountIdentified

5.1.4 Decision Model

The loan managers covered in this generic komponent vary in the following features:

- Reservation: support for reservations
- maxExtensions: the number of extensions a customer can get on a loaned item (integer value)
- loanPeriod: the length of a loan period (time value)

ID	Variation	Resolution	Effect
LM-S1	Reservation	yes (default)	yes: LM1.1, LM2.1, LM3.1, LM4.1, LM5.1, LM6.1, LM7.1, LM8.1, LM-R1
		no	no: LM1.1, LM2.1, LM3.1, LM4.1, LM5.1, LM6.1, LM7.1, LM8.1, LM-R1
LM-S2	loanPeriod	value [time]	replace loanPeriod by actual value: LM3.2, LM5.2
LM-S3	maxExtensions	value [int]	replace maxExtensions by actual value: LM5.3

Table 23: Integrated Decision Model for LoanManager Specification

ID	Variation	Resolution	Effect
LM1.1	Reservation	yes (default)	—
		no	remove method LoanManager.reserveItem() remove class ReservationList remove association class Reservation

Table 24: Decision Model for LoanManager Specification Class Diagram(Figure 29)

ID	Variation	Resolution	Effect
LM2.1	Reservation	yes (default)	—
		no	remove method reserveItem

Table 25: Decision Model for LoanManager Specification Supplied and Required Interfaces (Figure 30)

ID	Variation	Resolution	Effect
LM3.1	Reservation	yes (default)	—
		no	remove variant tags and content
LM3.2	loanPeriod	value [time]	replace loanPeriod by actual value

Table 26: Decision Model for LoanManager.loanItem() operation schema

ID	Variation	Resolution	Effect
LM4.1	Reservation	yes (default)	—
		no	remove variant tags and content

Table 27: Decision Model for LoanManager.returnItem() operation schema

ID	Variation	Resolution	Effect
LM5.1	Reservation	yes (default)	—
		no	remove variant tags and content
LM5.2	loanPeriod	value [time]	replace loanPeriod by actual value
LM5.3	maxExtensions	value [int]	replace maxExtensions by actual value

Table 28: Decision Model for LoanManager.reloanItem() operation schema

ID	Variation	Resolution	Effect
LM6.1	Reservation	yes (default)	—
		no	exclude operation specification from functional model

Table 29: Decision Model for LoanManager.reserveItem() operation schema

ID	Variation	Resolution	Effect
LM7.1	Reservation	yes (default)	—
		no	remove event reserveItem

Table 30: Decision Model for LoanManager Statechart Diagram (Figure 32)

ID	Variation	Resolution	Effect
LM8.1	Reservation	yes (default)	—
		no	remove method reserveItem()

Table 31: Decision Model for LoanManager Statechart Table

5.2 Realization

5.2.1 Structural Model

5.2.1.1 Class Diagram

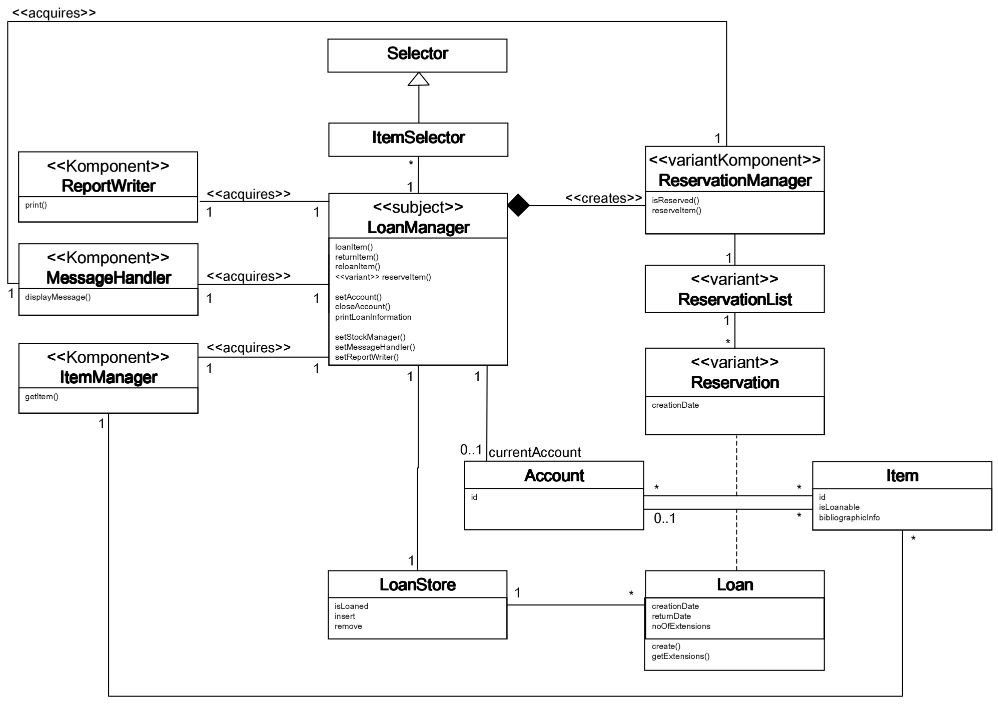


Figure 33: LoanManager Realization Class Diagram

5.2.1.2 Object Diagram

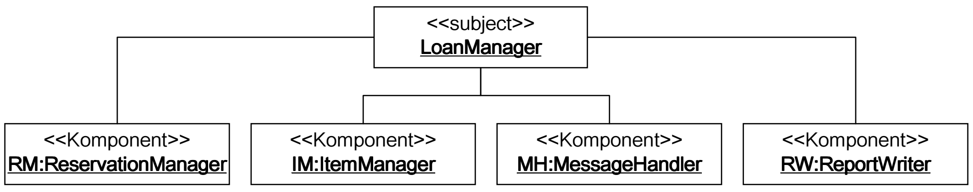


Figure 34: LoanManager Realization Object Diagram

5.2.2 Activity Model

5.2.2.1 Activity Diagram

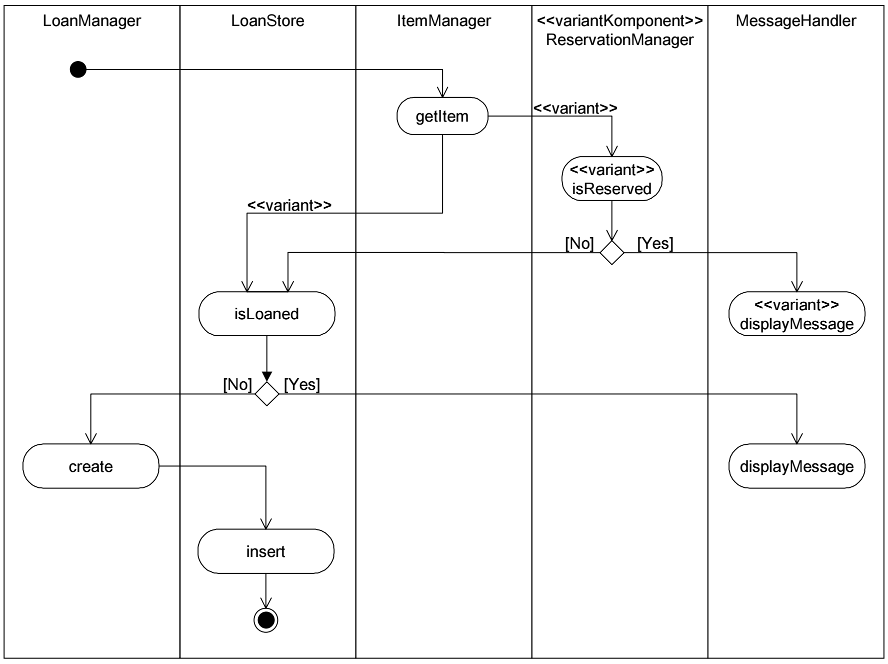


Figure 35: Activity Diagram for the loanItem Activity

5.2.3 Interaction Model

5.2.3.1 Collaboration Diagrams

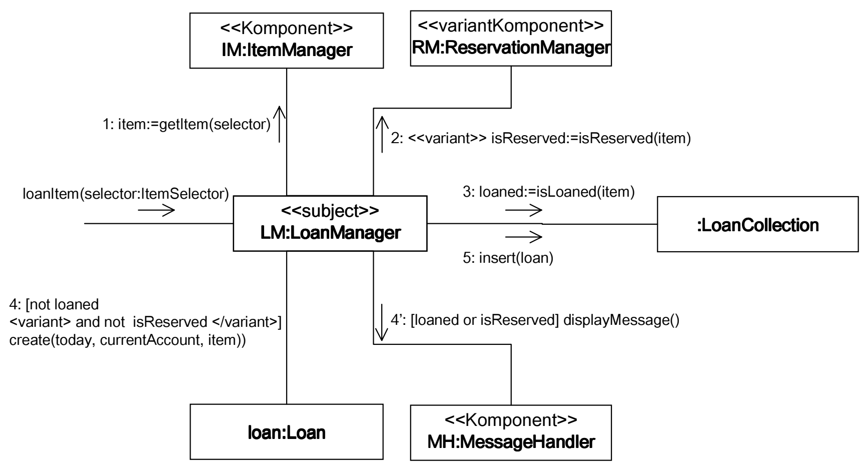

<<Komponent>>
IM:ItemManager
<<variantKomponent>>
RM:ReservationManager
1: item:=getItem(selector)
2: <<variant>> isReserved:=isReserved(item)
loanItem(selector:ItemSelector)
<<subject>>
LM:LoanManager
3: loaned:=isLoaned(item)
:LoanCollection
5: insert(loan)
4: [not loaned
<variant> and not isReserved </variant>]
create(today, currentAccount, item))
4': [loaned or isReserved] displayMessage()
loan:Loan
<<Komponent>>
MH:MessageHandler

Figure 36: Collaboration Diagram for the loanItem() Operation

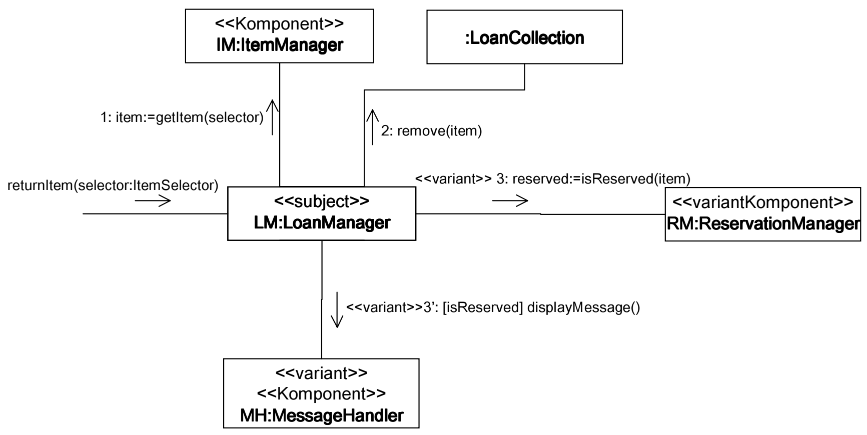
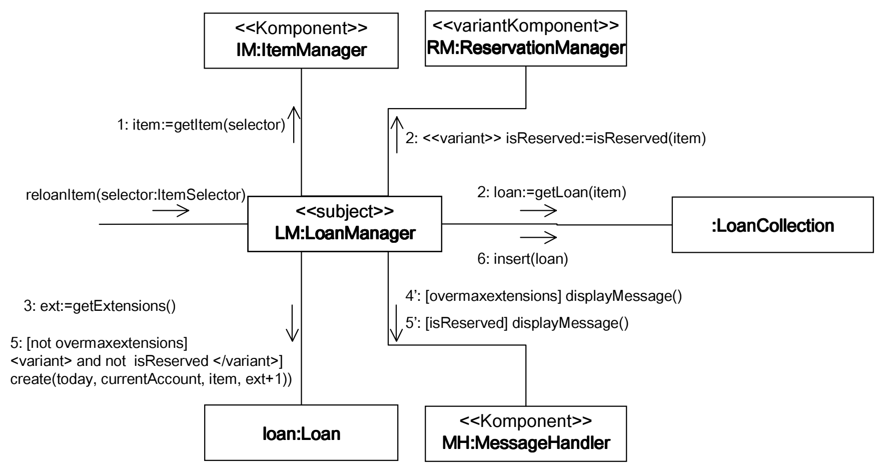
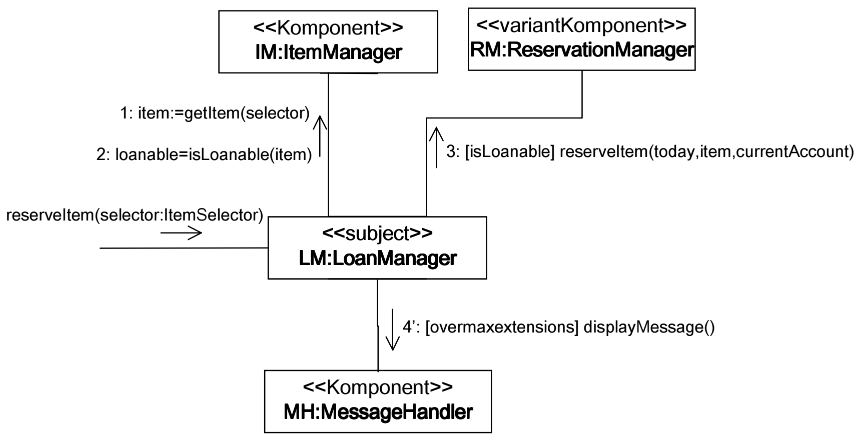


Figure 37: Collaboration Diagram for the returnItem() Operation



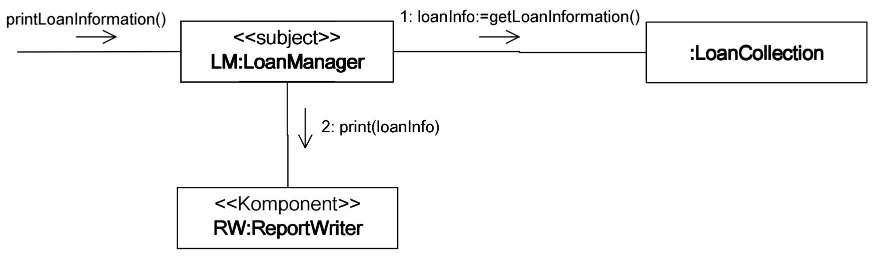
<<Komponent>> IM:ItemManager
<<variantKomponent>> RM:ReservationManager
1: item:=getItem(selector)
2: <<variant>> isReserved:=isReserved(item)
reloanItem(selector:ItemSelector)
<<subject>> LM:LoanManager
2: loan:=getLoan(item)
6: insert(loan)
:LoanCollection
3: ext:=getExtensions()
5: [not overmaxextensions] <variant> and not isReserved </variant>] create(today, currentAccount, item, ext+1))
4': [overmaxextensions] displayMessage()
5': [isReserved] displayMessage()
loan:Loan
<<Komponent>> MH:MessageHandler

Figure 38: Collaboration Diagram for the reloanItem() Operation



<<Komponent>> IM:ItemManager
<<variantKomponent>> RM:ReservationManager
1: item:=getItem(selector)
2: loanable=isLoanable(item)
3: [isLoanable] reserveItem(today,item,currentAccount)
reserveItem(selector:ItemSelector)
<<subject>> LM:LoanManager
4': [overmaxextensions] displayMessage()
<<Komponent>> MH:MessageHandler

Figure 39: Collaboration Diagram for the reserveItem() Operation



printLoanInformation()
<<subject>> LM:LoanManager
1: loanInfo:=getLoanInformation()
:LoanCollection
2: print(loanInfo)
<<Komponent>> RW:ReportWriter

Figure 40: Collaboration Diagram for the printLoanInformation() Operation

5.2.4 Decision Model

ID	Variation	Resolution	Effect
LM-R1	Reservation	yes (default)	yes: LM9.1, LM10.1, LM11.1, LM12.1, LM13.1, LM14.1
		no	no: LM9.1, LM10.1, LM11.1, LM12.1, LM13.1, LM14.1

Table 32: Integrated Decision Model for LoanManager Realization

ID	Variation	Resolution	Effect
LM9.1	Reservation	yes (default)	—
		no	remove method LoanManager.reserveItem() remove komponent ReservationManager remove class ReservationList remove association class Reservation

Table 33: Decision Model for LoanManager Realization Class Diagram(Figure 33)

ID	Variation	Resolution	Effect
LM10.1	Reservation	yes (default)	—
		no	remove swimlane ReservationManager remove activity isReserved remove activity displayMessage

Table 34: Decision Model for LoanManager.loanItem() Activity Diagram (Figure 35)

ID	Variation	Resolution	Effect
LM11.1	Reservation	yes (default)	—
		no	remove object RM:ReservationManager remove variant tags and content

Table 35: Decision Model for LoanManager.loanItem() Collaboration Diagram (Figure 36)

ID	Variation	Resolution	Effect
LM12.1	Reservation	yes (default)	—
		no	remove object RM:ReservationManager remove object MH:MessageHandler

Table 36: Decision Model for LoanManager.returnItem() Collaboration Diagram (Figure 37)

ID	Variation	Resolution	Effect
LM13.1	Reservation	yes (default)	—
		no	remove object RM:ReservationManager remove method call 5'[isReserved] displayMessage remove variant tags and content

Table 37: Decision Model for LoanManager.reloanItem() Collaboration Diagram (Figure 38)

ID	Variation	Resolution	Effect
LM14.1	Reservation	yes (default)	—
		no	exclude collaboration diagram from interaction model

Table 38: Decision Model for LoanManager.reserveItem() Collaboration Diagram (Figure 39)

6 Reservation Manager

6.1 Specification

6.1.1 Structural Model

6.1.1.1 Class Diagram

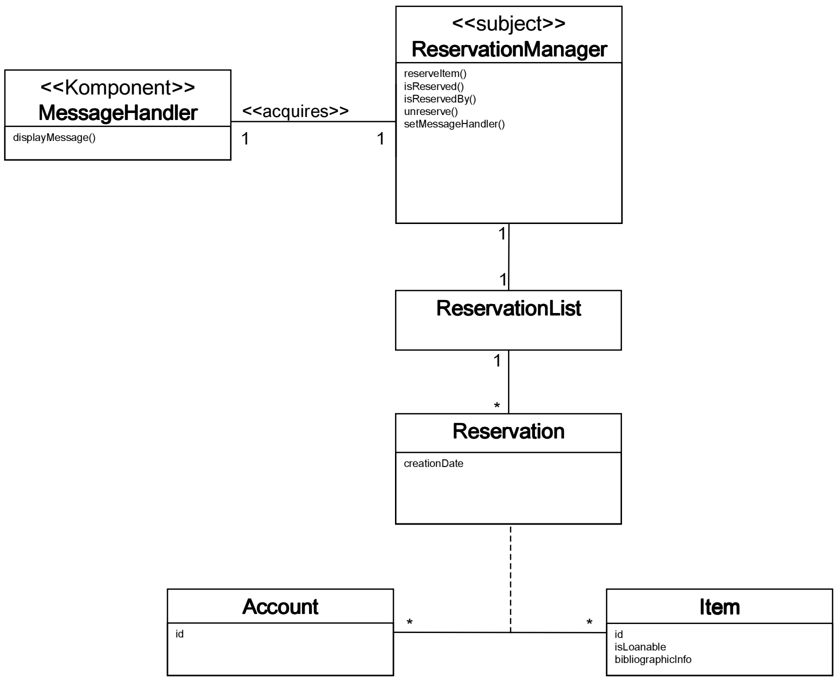


Figure 41: ReservationManager Specification Class Diagram

6.1.1.2 Supplied and Required Interfaces

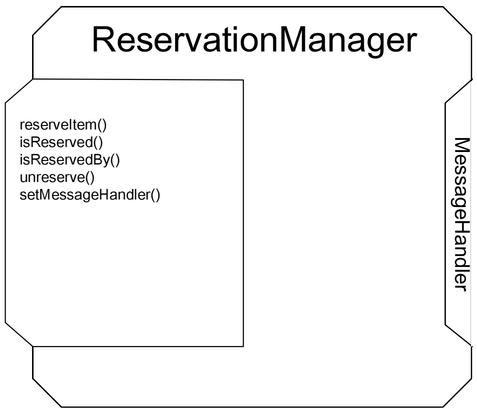


Figure 42: Supplied and Required Interfaces of the ReservationManager Komponent

6.1.1.3 Object Diagram

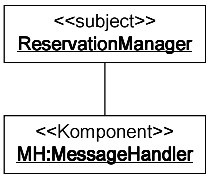


Figure 43: ReservationManager Specification Object Diagram

6.1.2 Functional Model

6.1.2.1 Operation Specifications

isReserved

Name	isReserved()
Description	Returns true or false depending on whether an item is reserved
Receives	item:Item
Sends	Boolean
Rules	—
Changes	—
Assumes	—
Result	If item is contained in one the reservations stored by ReservationManager True has been returned. Otherwise False is returned.

reserveItem

Name	reserveItem()
Description	Reserves an item for an account
Receives	item:Item account:Account
Sends	MH.displayMessage()
Rules	—
Changes	new Reservation
Assumes	—
Result	If a reservation of item for account already existed, the message displayMes-sage(“Already Reserved for Account”) has been sent to MH. Otherwise, a new reservation has been created relating item and account with attribute value - reservationDate=today

isReservedBy

Name	isReservedBy()
Description	It is checked whether an item is reserved and for which account
Receives	item:Item
Sends	account:Account
Rules	—
Changes	—
Assumes	—
Result	If item is contained in one the reservations stored by ReservationManager the account for which the item is reserved has been returned.

unreserve

Name	unreserve()
Description	Removes the reservation for an item and an account.
Receives	item:Item account:Account
Sends	—
Rules	—
Changes	destroys reservation
Assumes	There is a reservation for item and account.
Result	The reservation relating account and item has been destroyed.

setMessageHandler

Name	setMessageHandler()
Description	Receives a reference to a MessageHandler Komponent and stores it.

Receives	MH:MessageHandler
Sends	—
Rules	—
Changes	—
Assumes	—
Result	MH has been stored.

6.1.3 Behavioral Model

The ReservationManager Komponent does not have meaningful states. Therefore, the behavioral mode is empty.

6.1.4 Decision Model

The ReservationManager Komponent does not contain variability. Therefore, there is no decision model.

6.2 Realization

6.2.1 Structural Model

6.2.1.1 Class Diagram

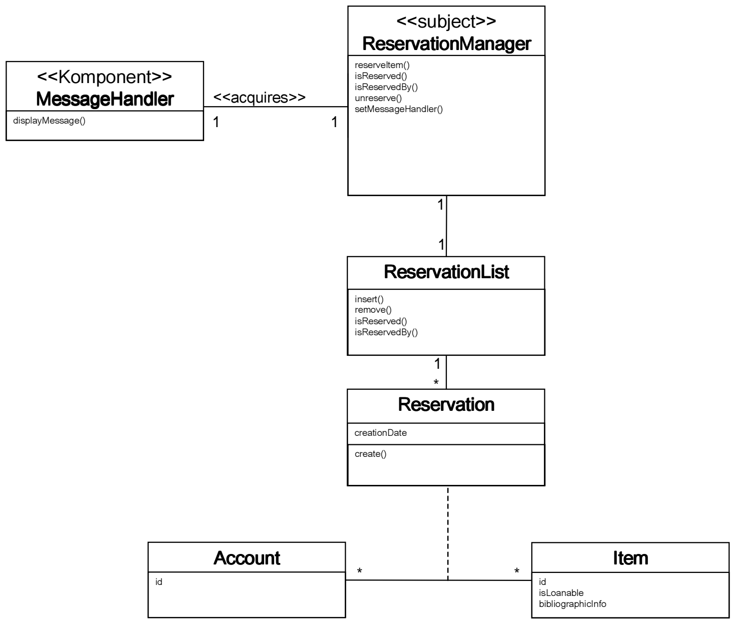


Figure 44: ReservationManager Realization Class Diagram

6.2.1.2 Object Diagram

Like the specification object diagram (see Figure 43).

6.2.2 Activity Model

6.2.2.1 Activity Diagrams

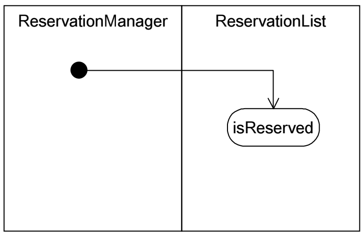


Figure 45: Activitity Diagram for the isReserved Activity

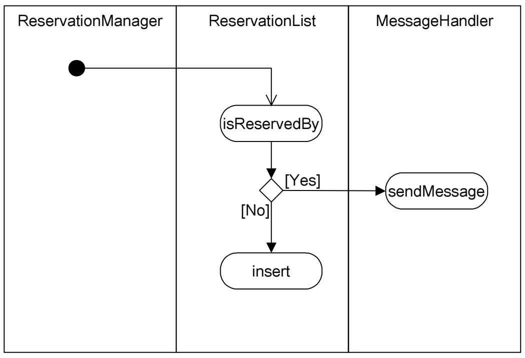


Figure 46: Activitity Diagram for the reserve Activity

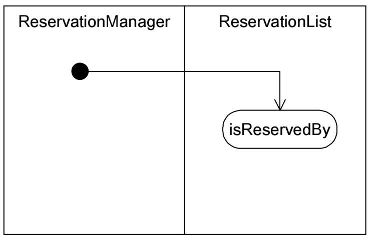


Figure 47: Activitity Diagram for the isReservedBy Activity

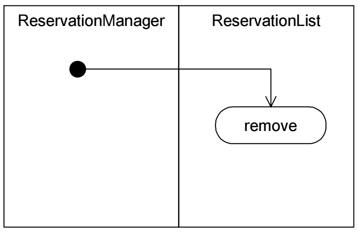


Figure 48: Activitity Diagram for

6.2.3 Interaction Model

6.2.3.1 Collaboration Diagrams

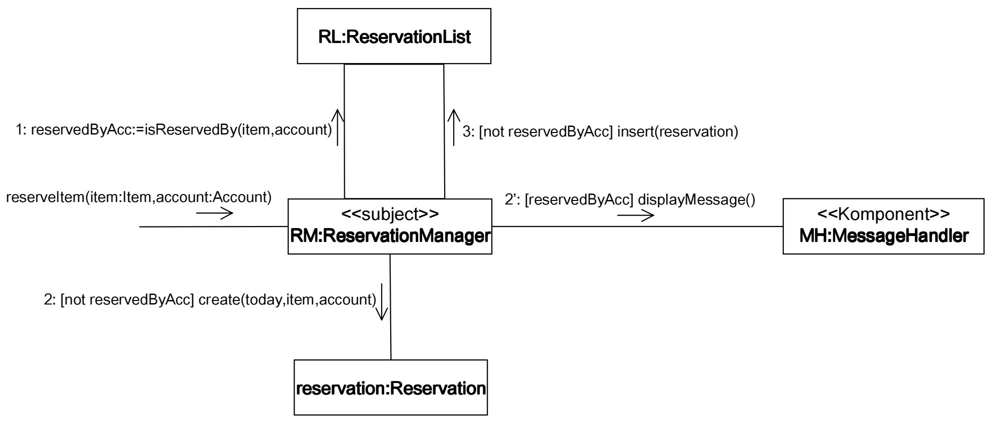

RL:ReservationList
1: reservedByAcc:=isReservedBy(item,account)
3: [not reservedByAcc] insert(reservation)
reserveItem(item:Item,account:Account)
<<subject>>
RM:ReservationManager
2': [reservedByAcc] displayMessage()
<<Komponent>>
MH:MessageHandler
2: [not reservedByAcc] create(today,item,account)
reservation:Reservation

Figure 49: Collaboration Diagram for reserveItem()

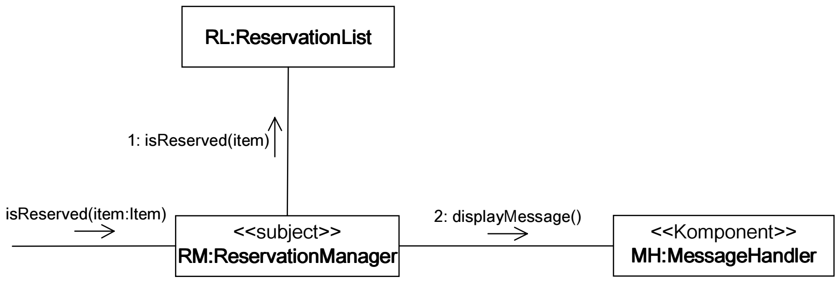


Figure 50: Collaboration Diagram for isReserved()

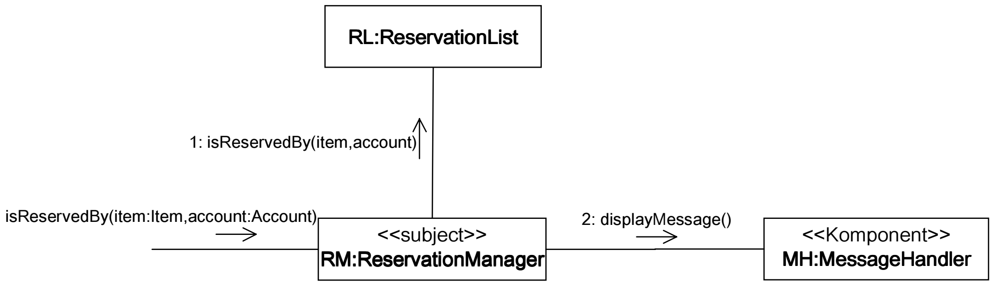


Figure 51: Collaboration Diagram for isReservedBy()

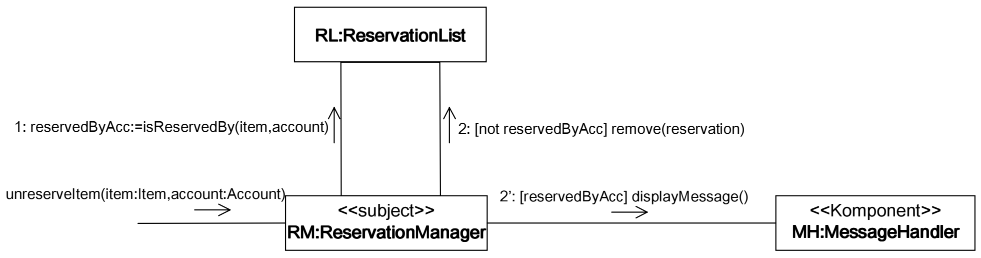

RL:ReservationList
1: reservedByAcc:=isReservedBy(item,account)
2: [not reservedByAcc] remove(reservation)
unreserveItem(item:Item,account:Account)
<<subject>>
RM:ReservationManager
2': [reservedByAcc] displayMessage()
<<Komponent>>
MH:MessageHandler

Figure 52: Collaboration Diagram for isReservedBy()

6.2.4 Decision Model

The ReservationManager Komponent does not contain variability. Therefore, there is no decision model.

Part II Basic Library System

7 Basic Library System

In this chapter the generic LibrarySystem presented in chapter 4 is instantiated for a basic library system. The basic library system communicates with an external database but does not support suggestion and reservation functionalities.

7.1 Specification

7.1.1 Structural Model

7.1.1.1 Class Diagram

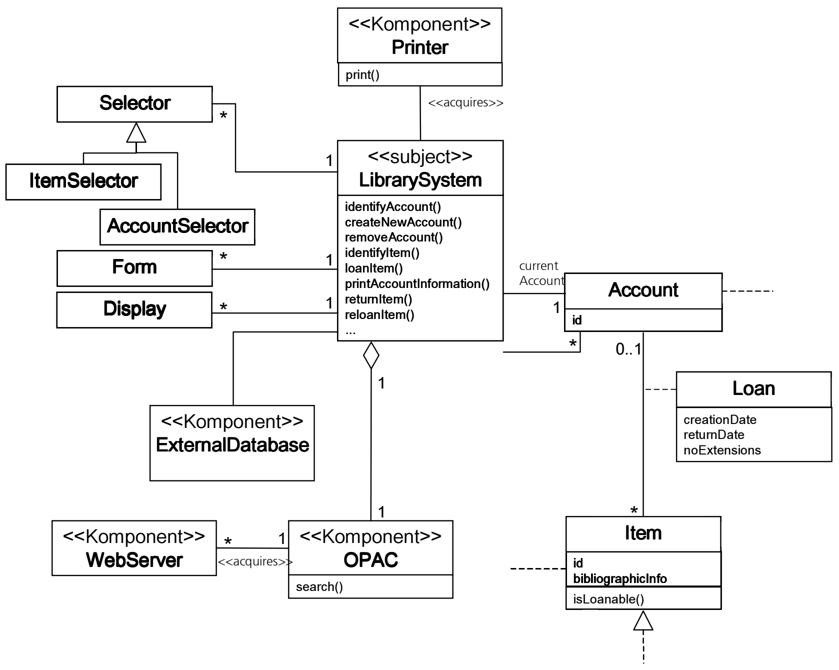


Figure 53: LibrarySystem Specification Class Diagram

7.1.1.2 Supplied and Required Interfaces

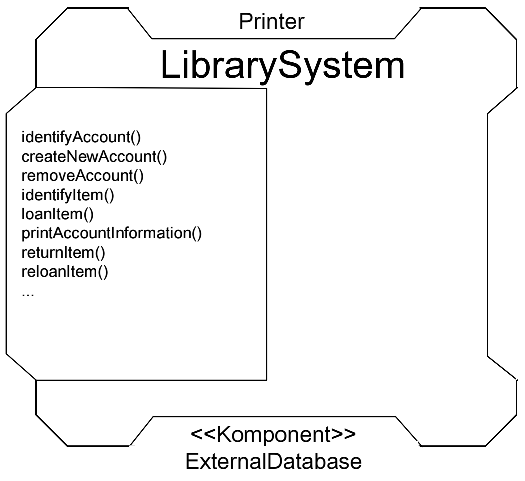


Figure 54: Supplied and Required Interfaces of the LibrarySystem Komponent

7.1.1.3 Object Diagram

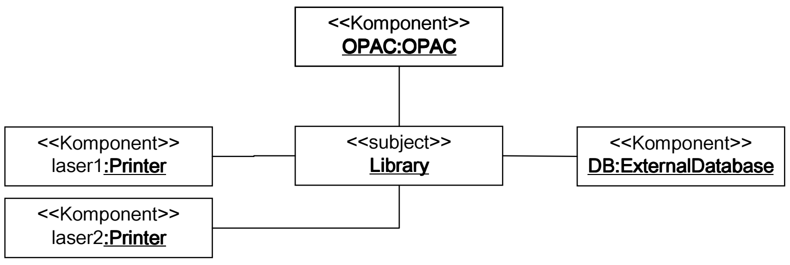


Figure 55: LibrarySystem Specification Object Diagram

7.1.2 Functional Model

7.1.2.1 Operation Specifications

loanItem

Name	loanItem()
Description	The loan of an Item to currentAccount is registered
Receives	selector: ItemSelector
Sends	Message "Already Loaned"
Rules	An item is loanable if it is not an item that must always stay in the library (e.g., antique books). An item is currently loanable if it is loanable and not loaned by another user.
Changes	new Loan
Assumes	Subject is in the state accountIdentified Selector selects exactly one Item
Result	item selected by selector has been obtained if item is currently loanable a new Loan object, loan, has been created that relates item and currentAccount and has the attribute values - creationDate = today - returnDate = today + 4 weeks and - noExtensions = 0 and, loan has been stored. if item is not currently loanable one of the messages has been displayed to the user "Already Loaned"

returnItem

Name	returnItem()
Description	Makes an item loanable again
Receives	selector:ItemSelector
Sends	
Rules	—
Changes	destroy loan
Assumes	subject is in the state accountIdentified selector selects exactly one item item is loaned to currentAccount
Result	item selected by selector has been obtained the loan for item and currentAccount has been destroyed

reloanItem

Name	reloanItem()
Description	An item loaned is reloaned to the currentAccount
Receives	selector:ItemSelector
Sends	Message “Over Extension”
Rules	An item is reloanable if it is loanable and the number of extension is less or equal to 5 .
Changes	loan
Assumes	Subject is in the state accountIdentified selector selects exactly one item Item is loaned to currentAccount
Result	item selected by selector has been obtained if item is reloanable the loan containing item has the attrbibute values - returnDate = today + 4 weeks - noExtensions = noExtensions+1 if item is not reloanable one of the following messages has been sent to MH displayMessage(“OverExtensions”)

printAccountInformation

Name	printAccountInformation()
Description	All information concerning the current account is printed.
Receives	—
Sends	Printer.print(data)
Rules	—
Changes	—
Assumes	subject is in state accountIdentified
Result	The data capturing customer data, current loans has been obtained, formatted, and sent to Printer.

createNewAccount

Name	createNewAccount()
Description	An account is created for a new customer
Receives	selector:AccountSelector
Sends	—
Rules	—
Changes	new Account
Assumes	selector does not select any existing account
Result	A new account:Account has been created according to the attributes of selector.

identifyAccount

Name	identifyAccount()
Description	An existing account is identified and opened.
Receives	selector:AccountSelector
Sends	—
Rules	—
Changes	currentAccount
Assumes	selector selects exactly one Account
Result	account:Account selected by selector has been obtained Library has been transitioned to state accountIdentified with currentAccount=account

removeAccount

Name	removeAccount()
Description	The currently selected account is closed and removed from the library
Receives	—
Sends	Message “Return all items first”
Rules	—
Changes	destroy loan
Assumes	subject is in the state accountIdentified
Result	If no Loans are related to currentAccount, currentAccount is destroyed.

7.1.3 Behavioral Model

7.1.3.1 Statechart Tables

In general, the behavioural model of systems of this size (or even bigger) is too complex for being captured in a useful state diagram. But if we hide the process support of the library system and focus only when which operations can be invoked, three main states of the library system can be identified:

- neutral: The system has no state information.
- accountIdentified: A particular account has been identified, which will be supplied to invoked operations (typically services with direct customer interaction with the library work via the customer account).
- itemIdentifed: A particular item has been identifed, which will be supplied to invoked operation (typically services in the background to maintain the library’s stock etc. are item-centric tasks)

The following statechart table lists for each of these three states the operations that can be invoked directly (i.e., not within a process execution) from a user.

No State Information	• identifyAccount • identifyItem • createNewAccount • …
accountIdentified	• removeAccount • loanItem • printAccountInformation • returnItem • reloanItem • …
itemIdentified	• reportLoss • …

7.1.3.2 State Diagram

The statechart table can be translated into the form of a UML statechart diagram. Here, the state diagram illustrates the conceptual states of the library system.

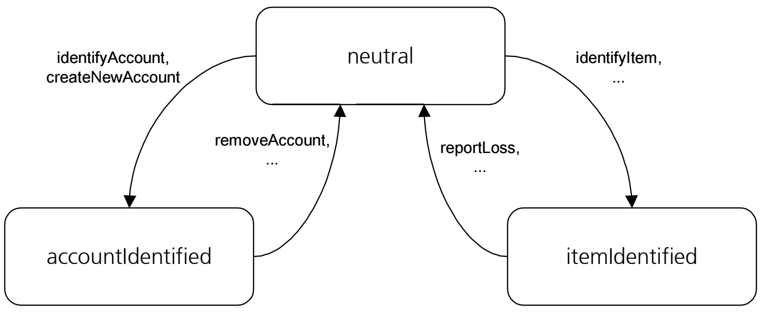


Figure 56:
State Transition Diagram: The main states of a library system

7.1.4 Resolution Model

In the scope definition given in the second chapter of this report, the considered library systems covered various varying concepts. Remember that in this report, we focus on variability related to features handling loaning and reserving items in a library. The generic specification of the komponent LibrarySystem varies in the following features:

- Reservation: support for reservations
- External Database: support for data ecxhange with an external database
- Suggestion: support for suggestions

- maxExtensions: the number of extensions a customer can get on a loaned item (integer value)
- loanPeriod: the length of a loan period (time value)

ID	Variation	Resolution	Effect
LS-S1	Reservation	no	no: LS1.1, LS2.1, LS4.1, LS5.1, LS6.1, LS7.1, LS8.1, LS11.1, LS12.1, LS-R1
LS-S2	External Database	yes	yes: LS1.2, LS2.2, LS3.1, LS12.1, ..., LS-R2
LS-S3	Suggestion	no	no: LS1.3, LS2.3, LS12.2, ..., LS-R3
LS-S4	loanPeriod	4 weeks	replace loanPeriod by actual value: LS4.2, LS6.2
LS-S5	maxExtensions	5	replace maxExtensions by actual value: LS6.3

Table 39: Integrated Resolution Model for LibrarySystem Specification

ID	Variation	Resolution	Effect
LS1.1	Reservation	no	remove method LibrarySystem.reserveItem() remove association class Reservation
LS1.2	External Database	yes	—
LS1.3	Suggestion	no	remove class Suggestion remove method LibrarySystem.suggestItem()

Table 40: Resolution Model for LibrarySystem Specification Class Diagram(Figure 53)

ID	Variation	Resolution	Effect
LS2.1	Reservation	no	remove method reserveItem()
LS2.2	External Database	yes	—
LS2.3	Suggestion	no	remove method suggestItem()

Table 41: Resolution Model for LibrarySystem Specification Supplied and Required Interfaces (Figure 54)

ID	Variation	Resolution	Effect
LS3.1	External Database	yes	—

Table 42: Resolution Model for LibrarySystem Specification Object Diagram (Figure 55)

ID	Variation	Resolution	Effect
LS4.1	Reservation	no	remove variant tags and content
LS4.2	loanPeriod	4 weeks	replace loanPeriod by actual value

Table 43: Resolution Model for LibrarySystem.loanItem() operation schema

ID	Variation	Resolution	Effect
LS5.1	Reservation	no	remove variant tags and content

Table 44: Resolution Model for LibrarySystem.returnItem() operation schema

ID	Variation	Resolution	Effect
LS6.1	Reservation	no	remove variant tags and content
LS6.2	loanPeriod	4 weeks	replace loanPeriod by actual value
LS6.3	maxExtensions	5	replace maxExtensions by actual value

Table 45: Resolution Model for LibrarySystem.reloanItem() operation schema

ID	Variation	Resolution	Effect
LS7.1	Reservation	no	exclude operation specification from functional model

Table 46: Resolution Model for LibrarySystem.reserveItem() operation schema

ID	Variation	Resolution	Effect
LS8.1	Reservation	no	remove variant tags and content in result clause

Table 47: Resolution Model for LibrarySystem.printAccountInformation() operation schema

ID	Variation	Resolution	Effect
LS11.1	Reservation	no	remove variant tags and content in result clause

Table 48: Resolution Model for LibrarySystem.removeAccount() operation schema

ID	Variation	Resolution	Effect
LS12.1	Reservation	no	remove method reserveItem()
LS12.1	External Database	yes (default)	—
LS12.2	Suggestion	no	remove method suggestItem()

Table 49: Resolution Model for LibrarySystem Statechart Table

7.2 Realization

The Realization of LibrarySystem focuses here on services that LibrarySystem assigns to the subkomponent LoanManager. Functionality that is aquired by other subcomponents of LibrarySystem from LoanManager is not taken into account.

7.2.1 Structural Model

7.2.1.1 Class Diagram

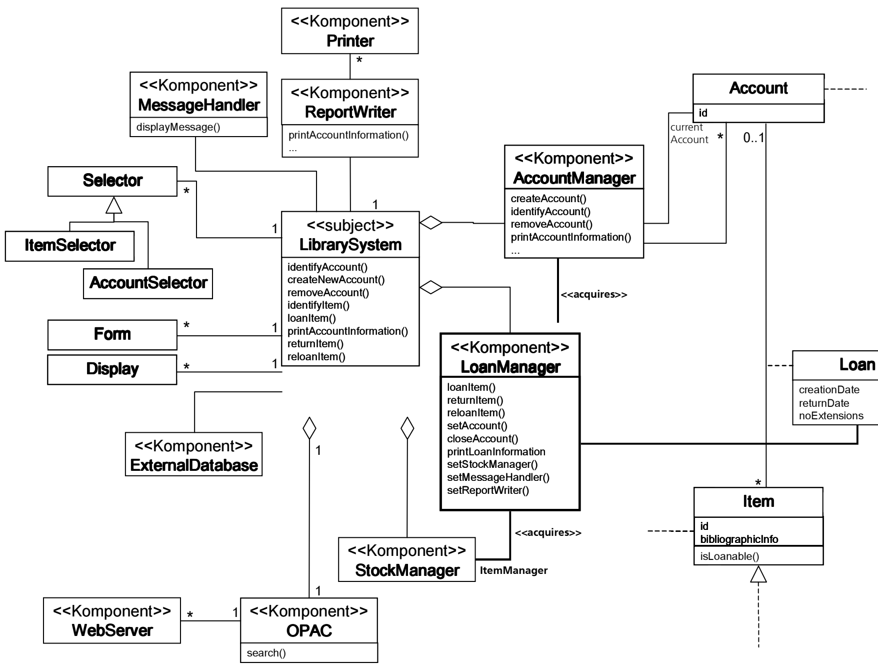


Figure 57: Library Realization Class Diagram

7.2.1.2 Object Diagram

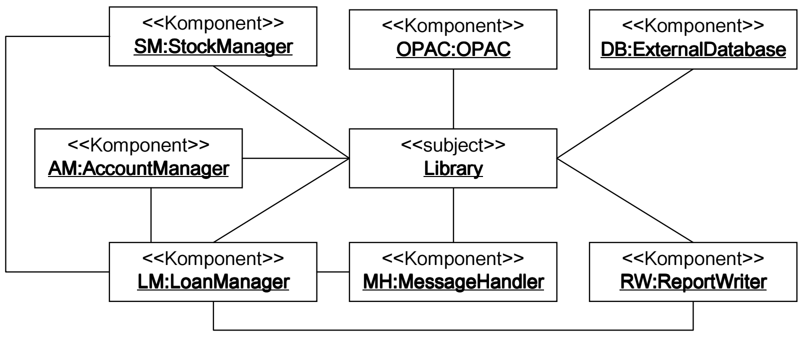


Figure 58: Library Realization Object Diagram

7.2.2 Activity Model

The simplicity of the activities taken into account in this report (i.e., activities related to loaning and reserving items) allowed us to realize them without intermediate refinement steps.

7.2.3 Interaction Model

In general, there are three ways for a subkomponent to be involved in the realization of its parent komponent's services:

- Delegation: a parent komponent does not add anything to the services provided by a subkomponent
- Synchronization: the state of the parent komponent changes in a way that requires a state change of the subkomponent to keep the system consistent.
- Usage: the parent komponent integrates numberous services of subkomponents to realize its (more powerful) services

7.2.3.1 Collaboration Diagrams

In our case of the loan and reservation functionality, the LibrarySystem fully delegates these services to a subkomponent (i.e., LoanManager)

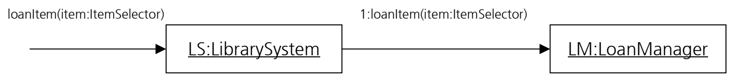


Figure 59: Collaboration Diagram for the loanItem() Operation

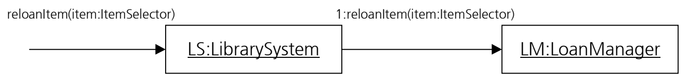

reloanItem(item:ItemSelector)
1:reloanItem(item:ItemSelector)
LS:LibrarySystem
LM:LoanManager

Figure 60: Collaboration Diagram for the reloanItem() Operation

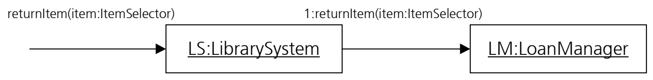

returnItem(item:ItemSelector)
1:returnItem(item:ItemSelector)
LS:LibrarySystem
LM:LoanManager

Figure 61: Collaboration Diagram for the returnItem() Operation

The LoanManager is not a stateless component but requires to specify an account first (currentAccount) which is then subject of subsequent actions. The AccountManager is responsible for Accounts independent of the LonaManager, therefore, some usages of AccountManager by the LibrarySystem requires an explicit state synchronization with the LoanManager komponent.

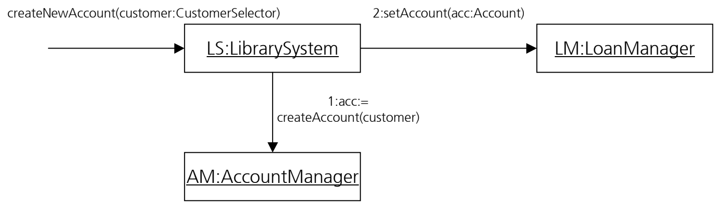


Figure 62: Collaboration Diagram for the createNewAccount() Operation

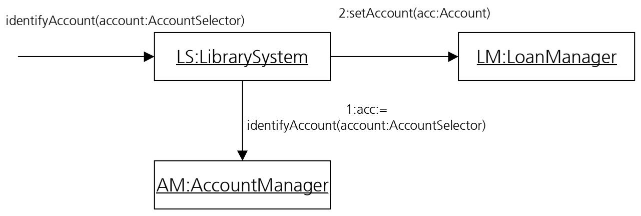

identifyAccount(account:AccountSelector)
2:setAccount(acc:Account)
LS:LibrarySystem
LM:LoanManager
1:acc:=
identifyAccount(account:AccountSelector)
AM:AccountManager

Figure 63: Collaboration Diagram for the identifyAccount() Operation

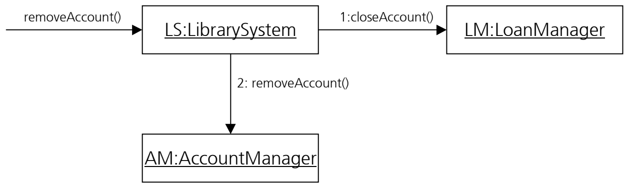


Figure 64: Collaboration Diagram for the removeAccount() Operation

The LibrarySystem allows account information to be printed in a single report. This information is spread over two of its subkomponents: AccountManager and

LoanManager. Therefore, to provide the service of printing account information, LibrarySystem must use and coordinate services of its subkomponents.

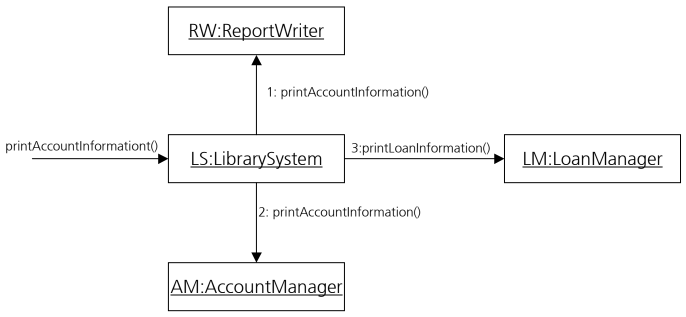


Figure 65: Collaboration Diagram for the printAccountInformation() Operation

7.2.4 Resolution Model

ID	Variation	Resolution	Effect
LS-R1	Reservation	no	no: LS13.1, LS15.1
LS-R2	External Database	yes	yes: LS13.2, LS14.1, ...
LS-R3	Suggestion	no	no: LS13.3, ...

Table 50: Integrated Resolution Model for LibrarySystem Realization

ID	Variation	Resolution	Effect
LS13.1	Reservation	no	remove method LibrarySystem.reserveItem() remove association class Reservation
LS13.2	External Database	yes	—
LS13.3	Suggestion	no	remove class Suggestion remove method LibrarySystem.suggestItem()

Table 51: Resolution Model for LibrarySystem Realization Class Diagram(Figure 57)

ID	Variation	Resolution	Effect
LS14.1	External Database	yes	—

Table 52: Resolution Model for LibrarySystem Specification Object DIagram (Figure 58)

ID	Variation	Resolution	Effect
LS15.1	Reservation	no	exclude collaboration diagram from interaction model

Table 53: Resolution Model for LibrarySystem.reserveItem() Collaboration Diagram

8 Basic Loan Manager

In this chapter the Loan Manager Komponent for the basic library system is presented. According to the Basic Library System Komponent, it does not support reservations.

8.1 Specification

8.1.1 Structural Model

8.1.1.1 Class Diagram

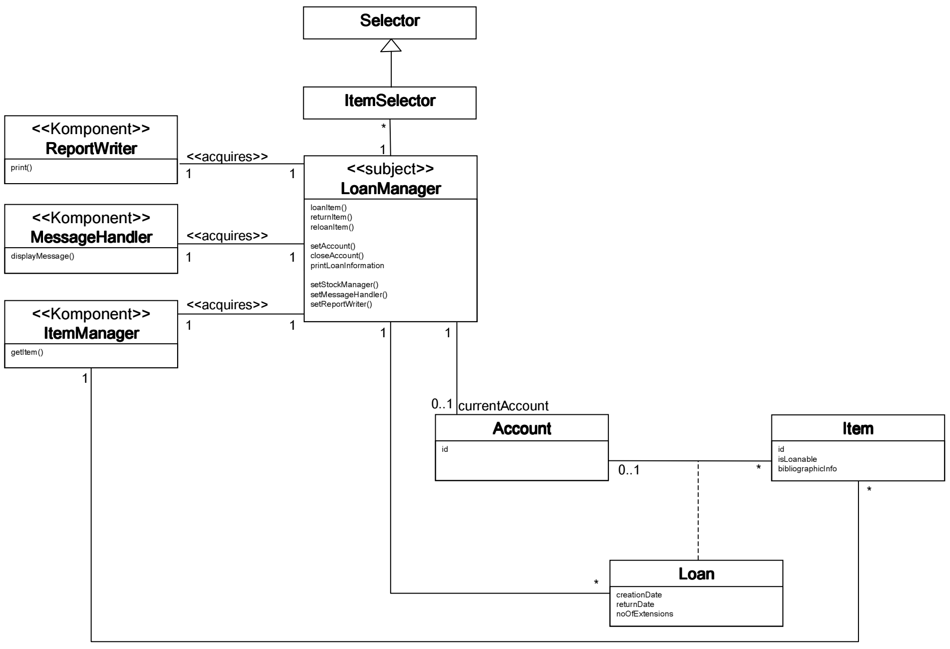


Figure 66: LoanManager Specification Class Diagram

8.1.1.2 Supplied and Required Interfaces

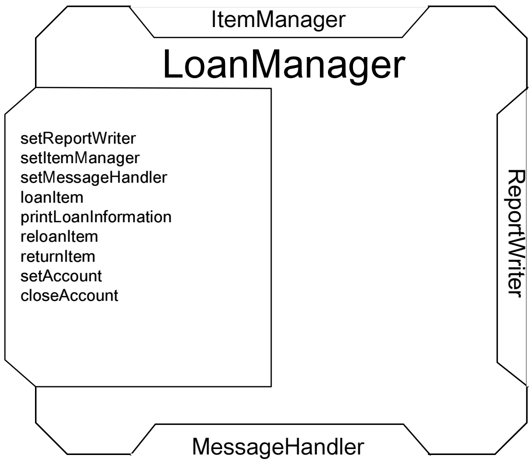


Figure 67: Supplied and Required Interfaces of the LoanManager Komponent

8.1.1.3 Object Diagram

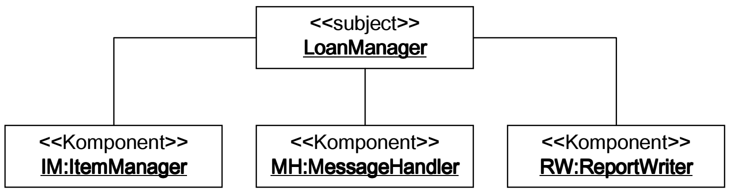


Figure 68: LoanManager Specification Object Diagram

8.1.2 Functional Model

8.1.2.1 Operation Specifications

setAccount

Name	setAccount()
Description	Receives a reference to an Account and stores it as currentAccount
Receives	currentAccount:Account
Sends	—
Rules	—
Changes	—
Assumes	—
Result	currentAccount has been stored and subject is in state accountIdentified

closeAccount

Name	closeAccount()
Description	The current account is closed.
Receives	—
Sends	—
Rules	—
Changes	—
Assumes	—
Result	currentAccount is empty and subject is in state noAccountIdentified

loanItem

Name	loanItem()
Description	The loan of an Item to currentAccount is registered
Receives	selector: ItemSelector
Sends	MH.displayMessage() Item item = IM.getItem(filter)
Rules	An item is loanable if it is not when it is not not an item that must always stay in the library (e.g., antique books). An item is currently loanable if it is loanable and not loaned.
Changes	new Loan
Assumes	Subject is in the state accountIdentified Selector selects exactly one Item
Result	item selected by selector has been obtained from the ItemManager IM by sending the message getItem(filter) if item is currently loanable a new Loan object, loan, has been created that relates item and account, and has the attribute values - creationDate = today - returnDate = today + 4 weeks and - noExtensions = 0 and, loan has been stored. if item is not currently loanable one of the messages has been sent to MH - displayMessage("Already Loaned")

returnItem

Name	returnItem()
Description	Makes an item loanable again.
Receives	selector:ItemSelector
Sends	MH.displayMessage() item Item = IM.getItem(selector)
Rules	—
Changes	destroy loan
Assumes	subject is in the state accountIdentified item is loaned to currentAccount
Result	item selected by selector has been obtained from the ItemManager IM by sending the message getItem(selector) the loan for item and currentAccount has been destroyed.

reloanItem

Name	reloanItem()
Description	An item loaned is reloaned to the currentAccount
Receives	selector:ItemSelector
Sends	MH.displayMessage() Item item = IM.getItem(selector)
Rules	An item is reloanable if it is loanable and the number of extension is less or equal to 5.
Changes	loan
Assumes	Subject is in the state accountIdentified Item is loaned to currentAccount
Result	item selected by selector has been obtained from the ItemManager IM by sending the message getItem(selector) if item is reloanable the loan containing item has the attrbibute values - returnDate = today + 4 weeks - noExtensions = noExtensions+1 if item is not reloanable one of the following messages has been sent to MH - displayMessage(“OverExtensions”)

printLoanInformation

Name	printLoanInformation()
Description	All information concerning loans of currentAccount is printed.
Receives	—
Sends	RW.printLoanInfo(currentAccount)
Rules	—
Changes	—
Assumes	subject is in state accountIdentified
Result	A message printLoanInfo(currentAccount) has been sent to RW.

setItemManager

Name	setItemManager()
Description	Receives a reference to an ItemManager komponent and stores it.
Receives	IM:ItemManager
Sends	
Rules	
Changes	
Assumes	
Result	IM has been stored

setMessageHandler

Name	setMessageHandler()
Description	Receives a reference to a MessageHandler komponent and stores it.
Receives	MH:MessageHandler
Sends	
Rules	
Changes	
Assumes	
Result	MH has been stored

setReportWriter

Name	setReportWriter()
Description	Receives a reference to a reportWriter komponent and stores it.
Receives	RW:ReportWriter
Sends	
Rules	
Changes	
Assumes	
Result	RW has been stored

8.1.3 Behavioral Model

8.1.3.1 Statechart Diagram

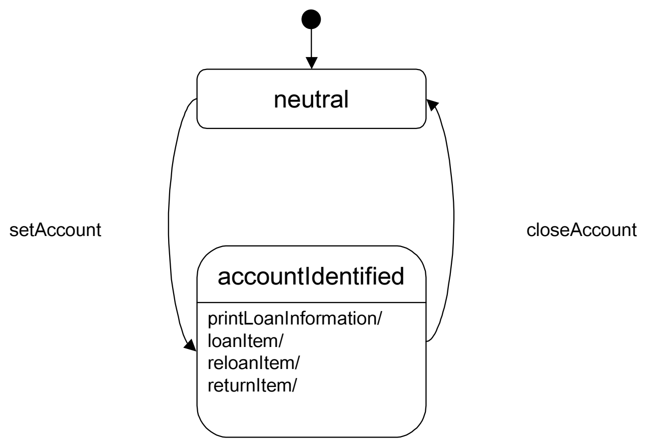


Figure 69: LoanManager Statechart Diagram

8.1.3.2 Statechart Tables

Source State	**UML Transition String**	**Target State**
neutral	setAccount()	accountIdentified
accountIdentified	closeAccount()	neutral

accountIdentified	printLoanInformation()
	loanItem()
	reloanItem()
	returnItem()
		accountIdentified

8.1.4 Decision Model

The loan managers covered in this generic komponent vary in the following features:

- Reservation: support for reservations
- maxExtensions: the number of extensions a customer can get on a loaned item (integer value)
- loanPeriod: the length of a loan period (time value)

ID	Variation	Resolution	Effect
LM-S1	Reservation	no	no: LM1.1, LM2.1, LM3.1, LM4.1, LM5.1, LM6.1, LM7.1, LM8.1, LM-R1
LM-S2	loanPeriod	4 weeks	replace loanPeriod by actual value: LM3.2, LM5.2
LM-S3	maxExtensions	5	replace maxExtensions by actual value: LM5.3

Table 54: Integrated Decision Model for LoanManager Specification

ID	Variation	Resolution	Effect
LM1.1	Reservation	no	remove method LoanManager.reserveItem() remove class ReservationList remove association class Reservation

Table 55: Decision Model for LoanManager Specification Class Diagram(Figure 66)

ID	Variation	Resolution	Effect
LM2.1	Reservation	no	remove method reserveItem()

Table 56: Decision Model for LoanManager Specification Supplied and Required Interfaces (Figure 67)

ID	Variation	Resolution	Effect
LM3.1	Reservation	no	remove variant tags and content
LM3.2	loanPeriod	4 weeks	replace loanPeriod by actual value

Table 57: Decision Model for LoanManager.loanItem() operation schema

ID	Variation	Resolution	Effect
LM4.1	Reservation	no	remove variant tags and content

Table 58: Decision Model for LoanManager.returnItem() operation schema

ID	Variation	Resolution	Effect
LM5.1	Reservation	no	remove variant tags and content

Table 59: Decision Model for LoanManager.reloanItem() operation schema

ID	Variation	Resolution	Effect
LM5.2	loanPeriod	4 weeks	replace loanPeriod by actual value
LM5.3	maxExtensions	5	replace maxExtensions by actual value

Table 59: Decision Model for LoanManager.reloanItem() operation schema

ID	Variation	Resolution	Effect
LM6.1	Reservation	no	exclude operation specification from functional model

Table 60: Decision Model for LoanManager.reserveItem() operation schema

ID	Variation	Resolution	Effect
LM7.1	Reservation	no	remove event reserveItem

Table 61: Decision Model for LoanManager Statechart Diagram (Figure 69)

ID	Variation	Resolution	Effect
LM8.1	Reservation		no

Table 62: Decision Model for LoanManager Statechart Table

8.2 Realization

8.2.1 Structural Model

8.2.1.1 Class Diagram

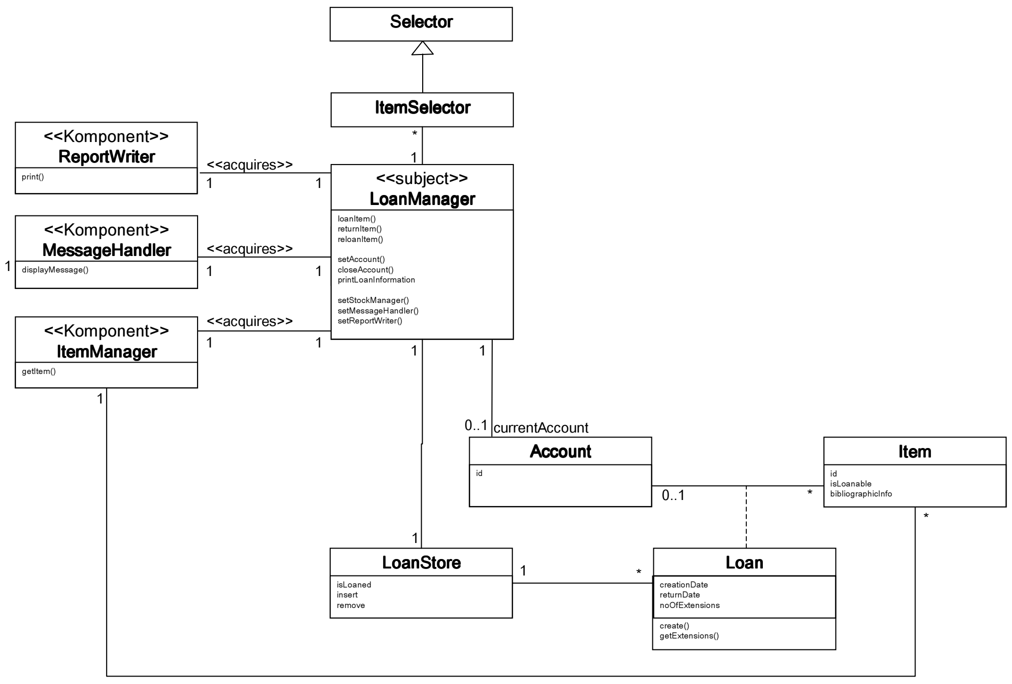


Figure 70: LoanManager Realization Class Diagram

8.2.1.2 Object Diagram

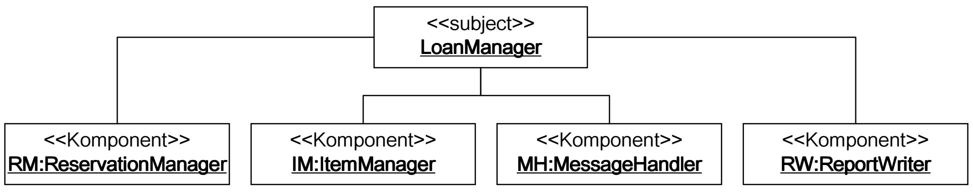

<<subject>> LoanManager
<<Komponent>> RM:ReservationManager
<<Komponent>> IM:ItemManager
<<Komponent>> MH:MessageHandler
<<Komponent>> RW:ReportWriter

Figure 71: LoanManager Realization Object Diagram

8.2.2 Activity Model

8.2.2.1 Activity Diagram

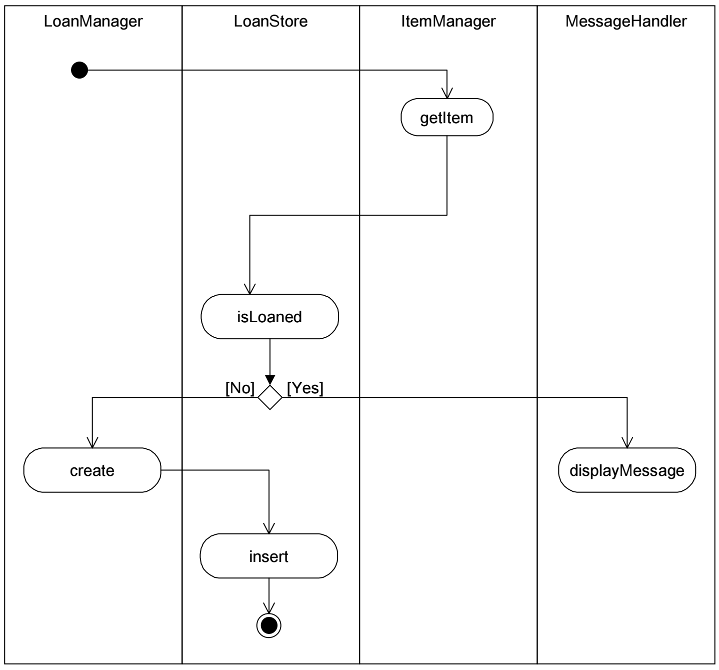
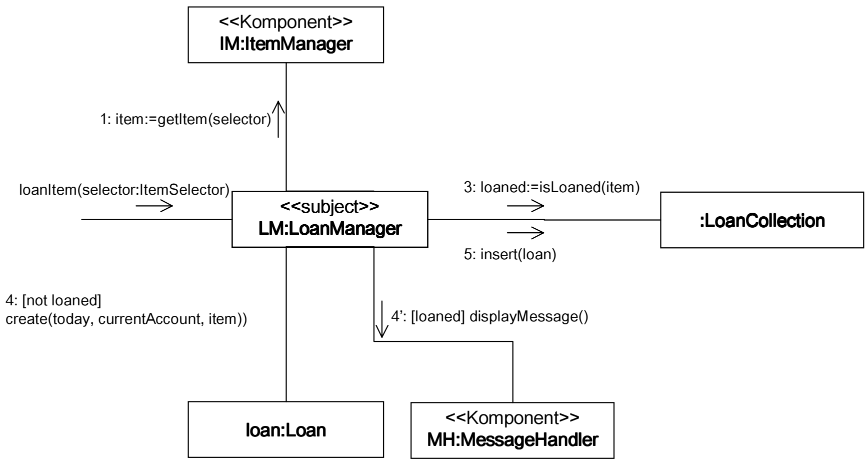


Figure 72: Activity Diagram for the loanItem Activity

8.2.3 Interaction Model

8.2.3.1 Collaboration Diagrams



<<Komponent>>
IM:ItemManager
1: item:=getItem(selector)
loanItem(selector:ItemSelector)
<<subject>>
LM:LoanManager
3: loaned:=isLoaned(item)
5: insert(loan)
:LoanCollection
4: [not loaned]
create(today, currentAccount, item))
4': [loaned] displayMessage()
loan:Loan
<<Komponent>>
MH:MessageHandler

Figure 73: Collaboration Diagram for the loanItem() Operation

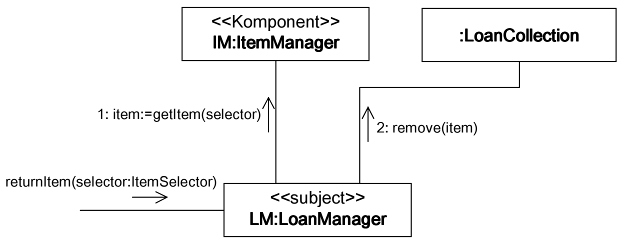


Figure 74: Collaboration Diagram for the returnItem() Operation

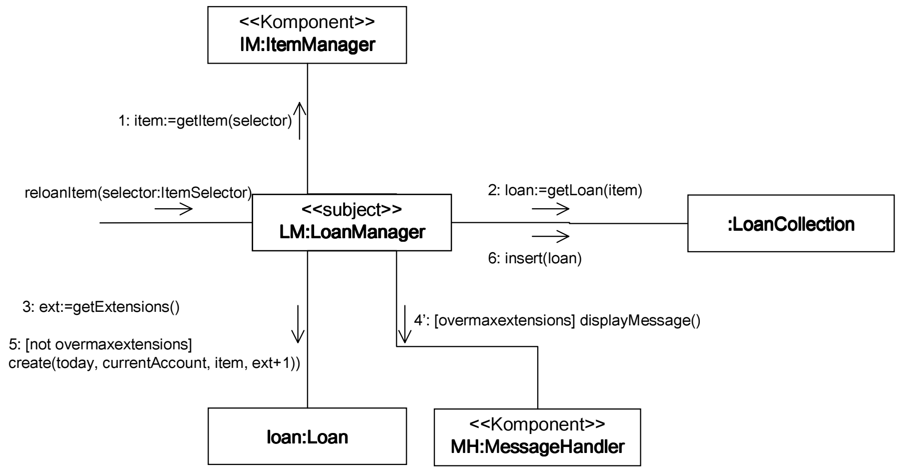

<<Komponent>>
IM:ItemManager
1: item:=getItem(selector)
reloanItem(selector:ItemSelector)
<<subject>>
LM:LoanManager
2: loan:=getLoan(item)
6: insert(loan)
:LoanCollection
3: ext:=getExtensions()
5: [not overmaxextensions]
create(today, currentAccount, item, ext+1))
4': [overmaxextensions] displayMessage()
loan:Loan
<<Komponent>>
MH:MessageHandler

Figure 75: Collaboration Diagram for the reloanItem() Operation

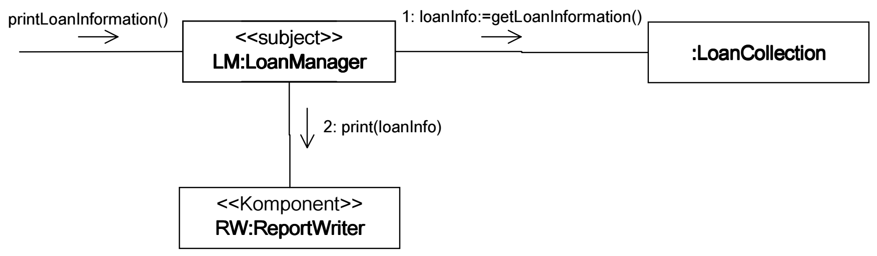


Figure 76: Collaboration Diagram for the printLoanInformation() Operation

8.2.4 Decision Model

ID	Variation	Resolution	Effect
LM-R1	Reservation	no	no: LM9.1, LM10.1, LM11.1, LM12.1, LM13.1, LM14.1

Table 63: Integrated Decision Model for LoanManager Realization

ID	Variation	Resolution	Effect
LM9.1	Reservation	no	remove method LoanManager.reserveItem() remove komponent ReservationManager remove class ReservationList remove association class Reservation

Table 64: Decision Model for LoanManager Realization Class Diagram(Figure 70)

ID	Variation	Resolution	Effect
LM10.1	Reservation	no	remove swimlane ReservationManager remove activity isReserved remove activity displayMessage

Table 65: Decision Model for LoanManager.loanItem() Activity Diagram (Figure 72)

ID	Variation	Resolution	Effect
LM11.1	Reservation	no	remove object RM:ReservationManager remove variant tags and content

Table 66: Decision Model for LoanManager.loanItem() Collaboration Diagram (Figure 73)

ID	Variation	Resolution	Effect
LM12.1	Reservation	no	remove object RM:ReservationManager remove object MH:MessageHandler

Table 67: Decision Model for LoanManager.returnItem() Collaboration Diagram (Figure 74)

ID	Variation	Resolution	Effect
LM13.1	Reservation	no	remove object RM:ReservationManager remove method call 5’[isReserved] displayMessage remove variant tags and content

Table 68: Decision Model for LoanManager.reloanItem() Collaboration Diagram (Figure 75)

ID	Variation	Resolution	Effect
LM14.1	Reservation	no	exclude collaboration diagram from interaction model

Table 69: Decision Model for LoanManager.reserveItem() Collaboration Diagram

References

[ABB+01] Colin Atkinson, Joachim Bayer, Christian Bunse, Erik Kamsties, Oliver Laitenberger, Roland Laqua, Dirk Muthig, Barbara Paech, Jürgen Wüst, and Jörg Zettel, Component-based Product Line Engineering with UML, Addison-Wesley, 2001.

[AW99] Marc Ardis and David Weiss, Defining Families: The Commonality Analysis. *Proceedings of the Nineteenth International Conference on Software Engineering*, pp. 649-650, IEEE Computer Society Press, May 1997.

[BFK+99] Joachim Bayer, Oliver Flege, Peter Knauber, Roland Laqua, Dirk Muthig, Klaus Schmid, Tanya Widen, and Jean-Marc Debaud, PuLSE: A Methodology to Develop Software Product Lines, In *Proceedings of the Symposium on Software Reusability 99 (SSR '99)*, Los Angeles, May 1999

[FODA98] Software Engineering Institute, Model-Based Software Engineering, http://www.sei.cmu.edu/technology/mbse/is.html, April 25, 1998.

[Hac92] Rupert Hacker. Bibliothekarisches Grundwissen - 6., völlig neu bearb. Aufl. - München [u.a.] : Saur, 1992

[PCM96] Meg Paul, Sandra Crabtree, Evelin Morgenstern, (Dt. Übers. u. Bearb.): Strategien für Spezialbibliotheken. Berlin : Deutsches Bibliotheksinstitut (DBI), 1996

[SPC93] Software Productivity Consortium. Reuse-Driven Software Processes Guidebook, Version 02.00.03. *Technical Report SPC-92019-CMC*, Software Productivity Consortium, November 1993.

Document Information

Title:	The Library System Product Line - A KobrA Case Study
Date:	November 22, 2001
Report:	IESE-024.01/E
Status:	Final
Distribution:	Public